Rajmohan Rajaraman
Thomas Moscibroda
Adam Dunkels
Anna Scaglione (Eds.)

LNCS 6131

Distributed Computing in Sensor Systems

6th IEEE International Conference, DCOSS 2010
Santa Barbara, CA, USA, June 2010
Proceedings

Springer

Rajmohan Rajaraman Thomas Moscibroda
Adam Dunkels Anna Scaglione (Eds.)

Distributed Computing in Sensor Systems

6th IEEE International Conference, DCOSS 2010
Santa Barbara, CA, USA, June 21-23, 2010
Proceedings

Springer

Volume Editors

Rajmohan Rajaraman
Northeastern University
College of Computer and Information Science (CCIS)
202 WVH, Boston, MA 02115, USA
E-mail: rraj@ccs.neu.edu

Thomas Moscibroda
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
E-mail: moscitho@microsoft.com

Adam Dunkels
Swedish Institute of Computer Science
Isafjordsgatan 22, 164 29, Kista, Sweden
E-mail: adam@sics.se

Anna Scaglione
University of California Davis
Department of Electrical and Computer Engineering
One Shields Avenue, Davis, CA 95616, USA
E-mail: ascaglione@ucdavis.edu

Library of Congress Control Number: 2010927968

CR Subject Classification (1998): C.2, H.4, D.2, C.2.4, F.2, H.3

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN 0302-9743
ISBN-10 3-642-13650-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-13650-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Message from the General Chair

We are pleased to present the proceedings of DCOSS 2010, the IEEE International Conference on Distributed Computing in Sensor Systems, the sixth event in this annual conference series. The DCOSS meeting series covers the key aspects of distributed computing in sensor systems, such as high-level abstractions, computational models, systematic design methodologies, algorithms, tools and applications.

We are greatly indebted to the DCOSS 2010 Program Chair, Rajmohan Rajaraman, for overseeing the review process and composing the technical program. We appreciate his leadership in putting together a strong and diverse Program Committee covering various aspects of this multidisciplinary research area.

We would like to thank the Program Committee Vice Chairs, Thomas Moscibroda, Adam Dunkels, and Anna Scaglione, as well as the members of the Program Committee, the external referees consulted by the Program Committee, and all of the authors who submitted their work to DCOSS 2010. We also wish to thank the keynote speakers for their participation in the meeting.

Several volunteers contributed significantly to the realization of the meeting. We wish to thank the organizers of the workshops collocated with DCOSS 2010 as well as the DCOSS Workshop Chair, Sotiris Nikoletseas, for coordinating workshop activities. We would like to thank Neal Patwari and Michael Rabbat for their efforts in organizing the poster and demonstration session. Special thanks to Chen Avin for handling conference publicity, to Animesh Pathak for maintaining the conference website, and to Zachary Baker for his assistance in putting together this proceedings volume. Many thanks also go to Germaine Gusthiot for handling the conference finances. We would like to especially thank Jose Rolim, DCOSS Steering Committee Chair. His invaluable input in shaping this conference series, making various arrangements and providing overall guidance are gratefully acknowledged.

Finally, we would like to acknowledge the sponsors of DCOSS 2010. Their contributions enabled this successful conference. The research area of sensor networks is rapidly evolving, influenced by fascinating advances in supporting technologies. We sincerely hope that this conference series will serve as a forum for researchers working in different, complementary aspects of this multidisciplinary field to exchange ideas and interact and cross-fertilize research in the algorithmic and foundational aspects, high-level approaches as well as more applied and technology-related issues regarding tools and applications of wireless sensor networks.

June 2010 Bhaskar Krishnamachari

Message from the Program Chair

This proceedings volume includes the accepted papers of the 6th International Conference on Distributed Computing in Sensor Systems. DCOSS 2010 received 76 submissions in three tracks covering the areas of algorithms, systems and applications. During the review procedure three (or more) reviews were solicited for all papers. After a fruitful exchange of opinions and comments at the final stage, 28 papers (36.8% acceptance ratio) were accepted.

The research contributions in this proceedings span diverse important aspects of sensor networking, including energy management, communication, coverage and tracking, time synchronization and scheduling, new programming paradigms, medium access control, sensor deployment, data security, and mobility. A multitude of novel algorithmic design and analysis techniques, systematic approaches and application development methodologies are proposed for distributed sensor networking, a research area in which complementarity and cross-fertilization are of vital importance.

I would like to thank the three Program Vice-Chairs, Thomas Moscibroda (Algorithms), Adam Dunkels (Systems and Applications), and Anna Scaglione (Signal Processing and Information Theory) for agreeing to lead the review process in their track and for an efficient and smooth cooperation. I would also like to thank the members of the strong and broad DCOSS 2010 Program Committee, as well as the external reviewers who worked with them. I wish to thank the Steering Committee Chair Jose Rolim and the DCOSS 2010 General Chair Bhaskar Krishnamachari for their trust and valuable contributions in organizing the conference, as well as the Proceedings Chair, Zachary Baker, for his tireless efforts in preparing these conference proceedings.

June 2010 Rajmohan Rajaraman

Organization

General Chair

Bhaskar Krishnamachari	University of Southern California, USA

Program Chair

Rajmohan Rajaraman	Northeastern University, USA

Program Vice-Chairs

Algorithms and Performance Analysis

Thomas Moscibroda	Microsoft Research, USA

Systems and Applications

Adam Dunkels	Swedish Institute of Computer Science, Sweden

Signal Processing and Information Theory

Anna Scaglione	University of California at Davis, USA

Steering Committee Chair

Jose Rolim	University of Geneva, Switzerland

Steering Committee

Sajal Das	University of Texas at Arlington, USA
Josep Diaz	UPC Barcelona, Spain
Deborah Estrin	University of California, Los Angeles, USA
Phillip B. Gibbons	Intel Research, Pittsburgh, USA
Sotiris Nikoletseas	University of Patras and CTI, Greece
Christos Papadimitriou	University of California, Berkeley, USA
Kris Pister	University of California, Berkeley, and Dust, Inc., USA
Viktor Prasanna	University of Southern California, Los Angeles, USA

Poster and Demo Session Chairs

Neal Patwari	University of Utah, USA
Michael Rabbat	McGill University, Canada

Workshops Chair

Sotiris Nikoletseas	University of Patras and CTI, Greece

Proceedings Chair

Zachary Baker	Los Alamos National Lab, USA

Publicity Chair

Chen Avin	Ben Gurion University, Israel

Web Publicity Chair

Animesh Pathak	INRIA Paris-Rocquencourt, France

Finance Chair

Germaine Gusthiot	University of Geneva, Switzerland

Sponsoring Organizations

IEEE Computer Society Technical Committee on Parallel Processing (TCPP)
IEEE Computer Society Technical Committee on Distributed Processing (TCDP)

Held in Cooperation with

ACM Special Interest Group on Computer Architecture (SIGARCH)
ACM Special Interest Group on Embedded Systems (SIGBED)
European Association for Theoretical Computer Science (EATCS)
IFIP WG 10.3

Program Committee

Algorithms and Performance

Stefano Basagni	Northeastern University, USA
Alex Dimakis	USC, USA
Eric Fleury	INRIA, France
Jie Gao	Stony Brook University, USA
Rachid Guerraoui	EPFL, Switzerland
Indranil Gupta	UIUC, USA
Anupam Gupta	CMU, USA
Ed Knightly	Rice, USA
Kishore Kothapalli	IIIT Hyderabad, India
Li Erran Li	Bell Labs, USA
Mingyan Liu	University of Michigan, USA
Andrew McGregor	University of Massachussets Amherst, USA
Boaz Patt-Shamir	Tel Aviv University, Israel
Sriram Pemmaraju	University of Iowa, USA
Yvonne-Anne Pignolet	IBM, Switzerland
Dan Rubenstein	Columbia University, USA
Paolo Santi	Unversity of Pisa, Italy
Stefan Schmid	T-Labs Berlin, Germany
Aravind Srinivasan	University of Maryland, USA
Berthold Voecking	RWTH Aachen, Germany
Dorothea Wagner	KIT, Germany
Guoliang Xing	Michigan State University, USA
Haifeng Yu	University of Singapore, Singapore

Applications and Systems

Jan Beutel	ETH, Switzerland
Qing Cao	University of Tennessee, USA
Peter Corke	QUT, Australia
Kasun De Zoysa	University of Colombo, Sri Lanka
Stefan Dulman	TU Delft, The Netherlands
Lewis Girod	MIT, USA
Omprakash Gnawali	Stanford, USA
Olaf Landsiedel	KTH, Sweden
Luca Mottola	SICS, Sweden
Lama Nachman	Intel, USA
Edith Ngai	Uppsala University, Sweden
Bodhi Priyantha	Microsoft Research, USA
Michele Rossi	University of Padova, Italy
Antonio Ruzzelli	UCD, Ireland
Utz Roedig	University of Lancaster, UK
Thomas Schmid	UCLA, USA

Thanos Stathopoulus	Bell Labs, USA
Cormac Sreenan	UCC, Ireland
Nigramanth Sridhar	Cleveland State University, USA
Yanjun Sun	Texas Instruments, USA
Andreas Terzis	John Hopkins University, USA
Andreas Willig	TU Berlin, Germany

Signal Processing and Information

J. Francois Chamberland	Texas A&M , USA
Biao Chen	Syracuse University, USA
Mark Coates	McGill, Canada
Gianluigi Ferrari	University of Parma, Italy
Carlo Fischione	KTH, Sweden
John W. Fisher III	MIT, USA
Massimo Franceschetti	UCSD, USA
Martin Haenggi	University of Notre Dame, USA
Peter Y-W. Hong	NTHU, Taiwan
Tara Javidi	UCSD, USA
Vikram Krishnamurty	UBC, Canada
Tom Luo	UMN, USA
Urbashi Mitra	USC, USA
Yasamin Mostofi	UNM, USA
Angelia Nedic	UIUC, USA
Michael Rabbat	McGill, Canada
Bruno Sinopoli	CMU, USA
Youngschul Sung	KAIST, Republic of Korea
A. Kevin Tang	Cornell, USA
Parv Venkitasubramaniam	Lehigh University, USA
Venu Veravalli	UIUC, USA
Azadeh Vosoughi	University of Rochester, USA
Aaron Wagner	Cornell, USA

Referees

Ehsan Aryafar	Giancarlo Fortino	Stanislav Miskovic
Navid Azimi	Radhakrishna Ganti	Asal Naseri
Niels Browers	Anastasios Giannoulis	Michael O.Grady
Binbin Chen	Ryan Guerra	Boris Oreshkin
Yin Chen	Bastian Katz	Saurav Pandit
Geoff Coulson	JeongGil Ko	Paul Patras
Declan Delaney	O. Patrick Kreidl	Arash Saber
Mike Dinitz	Yee Wei Law	Rik Sarkar
Ian Downes	HyungJune Lee	Dennis Schieferdecker
Joshua Ellul	Gaia Maselli	Simone Silvestri

Konstantinos Tsianos	Markus Voelker	Junjie Xiong
Nicolas Tsiftes	Meng Wang	Yuan Yan
Deniz Ustebay	Zixuan Wang	Mehmet Yildiz
Sundaram Vanka	Kevin Wong	

Table of Contents

Tables: A Spreadsheet-Inspired Programming Model for Sensor Networks

James Horey[1], Eric Nelson[2], and Arthur B. Maccabe[1]

[1] Oak Ridge National Laboratory
{horeyjl,maccabe}@ornl.gov
[2] The Aerospace Corporation
eric.j.nelson@aero.org

Abstract. Current programming interfaces for sensor networks often target experienced developers and lack important features. Tables is a spreadsheet inspired programming environment that enables rapid development of complex applications by a wide range of users. Tables emphasizes ease-of-use by employing spreadsheet abstractions, including pivot tables and data-driven functions. Using these tools, users are able to construct applications that incorporate local and collective computation and communication. We evaluate the design and implementation of Tables on the TelosB platform, and show how Tables can be used to construct data monitoring, classification, and object tracking applications. We discuss the relative computation, memory, and network overhead imposed by the Tables environment. With this evaluation, we show that the Tables programming environment represents a feasible alternative to existing programming systems.

1 Introduction

End-user programming interfaces for sensor networks must be greatly improved. Currently, creating and managing applications for sensor networks is too complex for casual users. Many existing programming interfaces assume that end-users are expert programmers that prefer advanced techniques. Although collaborating with experts can relieve some of these problems, this is neither scalable or cost-effective. In addition, users may find that built-in functionality of current management tools is too limited. In order to facilitate the adoption of this technology by new users, sensor network programming interfaces must be usable by a wide array of users with varying programming experience while remaining flexible and powerful.

Adapting existing tools, such as relational databases and spreadsheets, and applying them to sensor networks has great potential and allows users to transfer their existing knowledge base and skills. The challenge associated with this approach is to limit certain interactions while keeping the interface flexible enough to create interesting applications. Query-based tools, including TinyDB [14] and Cougar [24] adapt the database model by employing an SQL-like language [6]. These techniques allow users to gather data and specify aggregation behavior. However, it can be difficult to express complex applications, such as object tracking, using these models.

Prior work in spreadsheet-inspired programming environments, including the author's initial work [13] and subsequent work by Woo et al. [23], demonstrate the challenges of adapting the spreadsheet model. For example, the work by Woo et al. lacks an

	TinyDB	*Tenet*	*Tables*
Syntax	SQL	Chained tasks	Spreadsheet functions
Local functions	Simple filters	Sampling, filters, statistics	Conditionals, arithmetic, statistics
Collective functions	Built-in functions	Arbitrary code on basestation	Conditionals, arithmetic, statistics
In-network aggregation	Yes	No	Yes
Simplified querying	Yes	No	Yes
Interactive use	Yes	No	Yes
Tiered architecture	Flexible	Rigid	Flexible

Fig. 1. Tables supports queries, functions, and collective functions with in-network aggregation

integrated programming model, thus limiting their interface to simple data collection. In order to successfully incorporate spreadsheet actions with sensor networks, we propose a new programming model based on data-driven functions and implicit communication between event-driven sensor groups. Data-driven functions are declarative statements that refer to named data, such as *Photometer* and can include arithmetic, mathematical, and conditional operators. These functions can be chained together by assigning data, similar to tasks in Tenet [10].

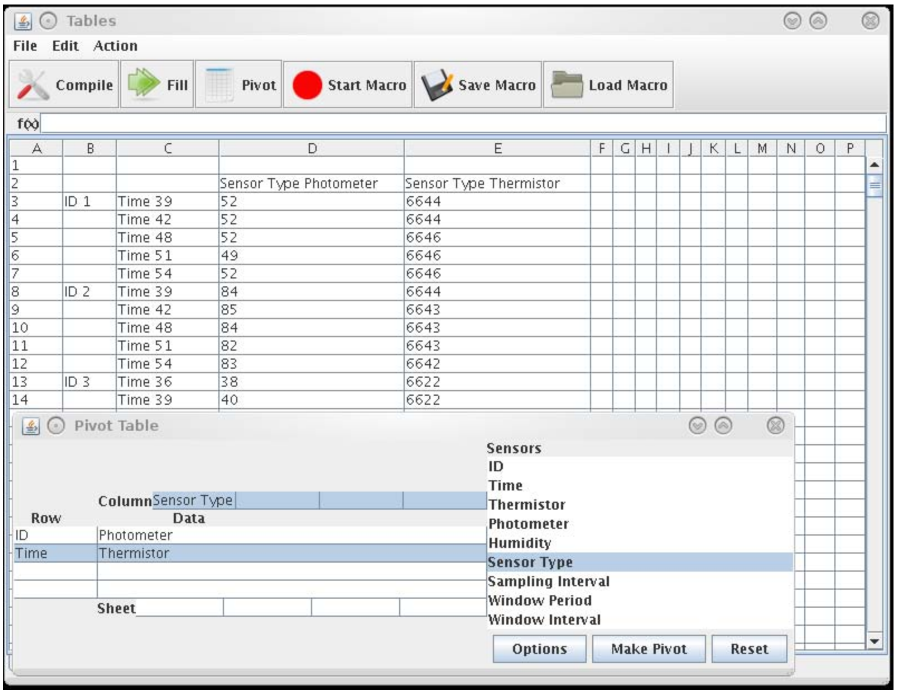
In addition, event-driven groups can be formed using the *pivot table* tool. This tool allows users to create sophisticated queries and visually organize data. With these tools, users can specify groups of nodes that share a common data constraint. Once these groups are created, data can be implicitly transferred between groups and the basestation to facilitate collective computation. Using this programming model, our software, Tables, combines the best aspects of both query-based and programmatic approaches (Figure 1).

In this paper, we present a novel programming model that integrates implicit tiered communication, event-driven sensor groups, and data-driven computation (Section 2). We discuss how this model can be integrated with spreadsheet-inspired tools and discuss our current prototype implementation on the TelosB platform (Section 3). Using these tools, we show how users can construct applications including periodic monitoring, data classification, in-network aggregation, and object tracking. We then analyze the memory, computational, and communication overheads of our prototype. Finally, we compare our system to related work (Section 4) and offer a brief conclusion (Section 5).

2 Programming Model

The main abstractions used by Tables are data-driven functions, event-based groups, and implicit, tiered communication. These abstractions are specified using a combination of spreadsheet-inspired tools. In order to support these abstractions, Tables excludes explicit data sampling and neighbor communication. In Tables, sensor data is automatically sampled and queued, although the user is able to modify the sampling periods. Although the lack of neighbor communication appears to be limiting, it is important to note that Tables does support in-network aggregation. Also, not including such features allows Tables to be potentially ported over non-networked sensors.

Because these programming abstractions are difficult to illustrate apart from the programmatic mechanisms, both mechanisms and abstractions will be discussed together. Two of the most important tools are the pivot table and the viewing area where the data is displayed. The viewing area resembles a typical spreadsheet: a two-dimensional table of cells. Each of these tables (aka: *sheet*), resides in a uniquely named tab, allowing the user to display data along three dimensions (row, column, and sheet).



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1															
2			Sensor Type Photometer	Sensor Type Thermistor											
3	ID 1	Time 39	52	6644											
4		Time 42	52	6644											
5		Time 48	52	6646											
6		Time 51	49	6646											
7		Time 54	52	6646											
8	ID 2	Time 39	84	6644											
9		Time 42	85	6643											
10		Time 48	84	6643											
11		Time 51	82	6643											
12		Time 54	83	6642											
13	ID 3	Time 36	38	6622											
14		Time 39	40	6622											

Column: Sensor Type

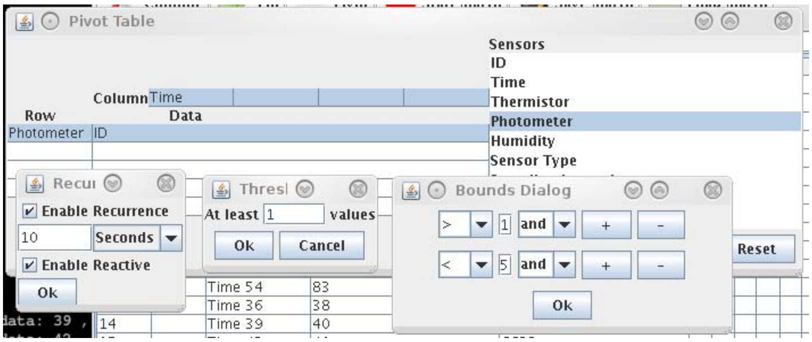
Row	Data
ID	Photometer
Time	Thermistor

Sheet

Sensors
ID
Time
Thermistor
Photometer
Humidity
Sensor Type
Sampling Interval
Window Period
Window Interval

Fig. 2. Pivot table to view Photometer and Thermistor data organized by Node ID and Time. Results are collected and reflect the pivot table organization.

The pivot table is a dialog that displays a miniature view of the spreadsheet. The dialog consists of a list of data names (aka: *sensor* list), a metadata pane for each spreadsheet axis, and a data pane. Items in the sensor list are populated with the names of sensor values, such as Photometer, and user assigned data. The user creates a query by dragging items from the sensor list onto one of the three metadata and data panes. Items contained in the metadata panes specify how the items in the data pane is to be organized. Each pane (with the exception of the sheet) is capable of containing multiple items, allowing users to create complex multi-dimensional queries.



☑ Enable Recurrence
10 Seconds
☑ Enable Reactive

At least 1 values

> 1 and
&lt; 5 and

Fig. 3. Pivot table to view ID organized by Photometer data with the threshold and constraint options. The pivot table is also configured to regularly transmit results every 10 seconds.

Once constructed, the pivot table is propagated to the sensor network and the results are displayed onto the viewing area. Figure 2 illustrates a pivot table requesting Photometer and Thermistor data organized by the node ID, timestamp, and the name of the data. For those more familiar with SQL syntax, this pivot table is similar to:

```
SELECT Thermistor, Photometer FROM Nodes
SORT BY ID, Time, Sensor Type
```

Unlike SQL, pivot tables have the additional advantage of organizing the data visually. Also, pivot tables are not limited to traditional sensor queries. It is just as easy to construct a pivot table to display node ID organized by Photometer data, etc. This is equivalent to the query:

```
SELECT ID FROM Nodes SORT BY Photometer
```

Users can also specify various options on pivot table items similar to the **WHERE** operator (Figure 3). The user can choose to display data within a certain range or specify a minimum number of elements that must be present. Because pivot tables normally execute immediately, the user can also create a *reactive* table. These tables are stored by sensor nodes until all the conditions are met. The user can also specify a *recurrence* so that the pivot table is reevaluated periodically. This option can be combined with reactive tables to create periodic pivot tables that occur only after the requirements have been met.

2.1 Data-Driven Functions

In addition to pivot tables, the user can also create data-driven functions that operate over sensor data. Unlike procedural functions, data-driven functions are only activated when data the function relies on is updated. For example, if a function refers to photometer data, the function will be executed when new photometer data is collected. Since all available sensor data is periodically collected, functions that rely on that data are also periodically evaluated. Unlike a normal spreadsheet function, functions in Tables must specifically refer to a data name instead of a cell reference. Currently, Tables

Functions	Examples
Arithmetic (+, -, *, /, sqrt)	*Photometer* + τ
Boolean (!, &, |, <, >, ==)	*Photometer* < τ & *Thermistor* > δ
Vector (min, max, sum, slope, average)	average(min values, *Photometer*)
Assignment	*a* := sqrt(*Photometer*)
Conditional	if(*Photometer* < τ) *a* := sqrt(*Photometer*)

Fig. 4. Tables supports arithmetic, boolean, and vector operations. Users refer to data elements using a string-based name and employ assignment and conditional functions to create new data.

supports arithmetic, boolean, several vector functions, and conditionals (Table 4). Functions can also be *chained* together using assignment functions (Figure 2.1). Although there are no explicit looping commands, recursive functions can be created using a conditional function that relies on its own data (Figure 5(b)).

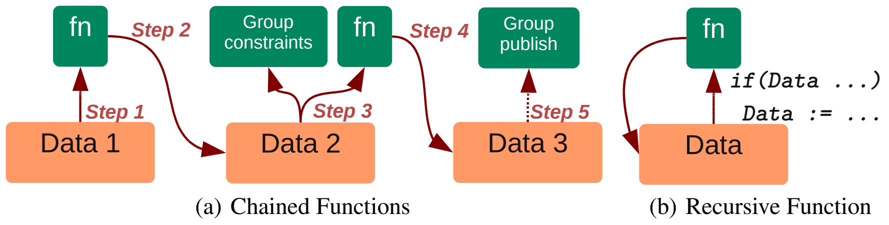


Fig. 5. Functions are associated with a data element and activated when new data arrives. Functions can be chained (a) and looped using a self-referential conditional function (b).

In order to create a new function, the user types the function into an empty cell. Initially the interface is populated with a sheet for every node in the network along with a non-constrained sheet. By placing a function in one of the node-specific sheets, the user explicitly tasks a particular sensor node with that function. If the function is placed in the non-constrained sheet, the function is tasked for *all* sensor nodes. Because these functions operate *locally*, data comes directly from the sensor node. For example, a function that manipulates Photometer data will get the data directly from the sensor node photometer queue. Similarly, if the function assigns a new value, the value is stored locally on the sensor node.

In combination with pivot tables, we envision these functions being used for both stand-alone data processing (ie: data filtering, classification, etc.) and in the context of more complex applications with both querying and collective elements. Because functions are data-driven, functions can be written independently allowing applications to be created piece-meal. Users can start out by only using pivot tables and later add local functions. This type of iterative interaction resembles the way typical spreadsheets are constructed and encourages experimentation.

2.2 Event-Based Groups

Users are also able to construct collective functions that operate over data from multiple sensor nodes. In order to create a collective function, the user must define a data-constrained sheet. Unlike node-constrained sheets, a data-constrained sheet is associated with a data element and a value. Once defined, the sheet represents *all* sensor nodes where the data element is *equal* to the specified value. For example, if a sheet has the data constraint “Photometer 100”, then all nodes where the latest Photometer value is equal to 100 will be represented by that sheet.

The user can create a data-constrained sheet manually or by using a pivot table. For pivot tables, the user drags a data element from the sensor list onto the sheet pane. This implicitly creates a set of data-constrained sheets. After the pivot table is created, all unique values associated with that data element will be associated with a sheet. For example, if the user specified *User Value* as the sheet data element, and assuming every node recorded a *User Value* of 1 or 2, the result will be a data-constrained sheet for 1 and 2.

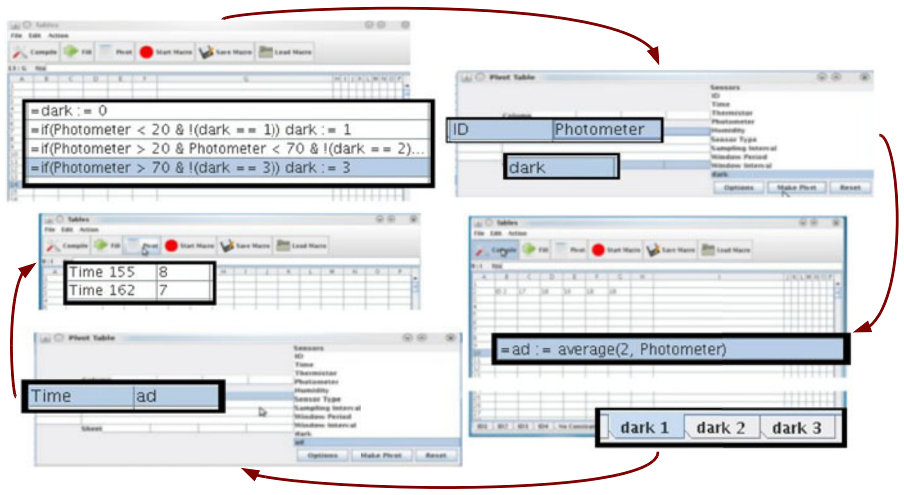
After creating a data-constrained sheet, the user can type in a function on the sheet. Unlike local functions, these functions will operate over data from multiple nodes. When possible Tables will perform in-network aggregation. Because a sheet represents a subset of sensor nodes, only nodes that satisfy the data constraint will transmit data for the aggregation. In combination with in-network aggregation, this can greatly reduce the number of messages transmitted ($40 - 50\%$ in many cases). Although our system currently executes collective functions on the basestation, the Tables programming model does not preclude alternative implementations. We hope to explore alternative implementations that focus on executing collective functions in the network in the near future.

2.3 Convenience Functions

In order to facilitate integration with other tools, Tables supports both a *macro* mode and *csv* output. Macro mode is initiated when the user clicks on the macro icon. All actions, including pivot tables and functions, are recorded. Later after the user saves the macro, the user can replay the macro either via the GUI or command-line interface. In either mode, interactive or macro, results from pivot tables are stored in comma-separated-variable files. This is done transparently without the user’s explicit input. These features can be used to interactively design applications and deploy them at future times.

2.4 Applications: Weather Classification and Object Tracking

Using a combination of pivot tables, local functions, and collective functions, the user can create applications that are difficult to express in other query-based languages. Two such applications, weather classification and object tracking, are straightforward to express in Tables. For weather classification, the goal is to organize the sensor nodes into three groups: one for *dark*, *dim*, and *light* areas and to examine the average photometer value in a particular group. The user starts by typing the following local functions into a non-constrained sheet.



```
=dark := 0
=if(Photometer < 20 & !(dark == 1)) dark := 1
=if(Photometer > 20 & Photometer < 70 & !(dark == 2))...
=if(Photometer > 70 & !(dark == 3)) dark := 3
```

ID	Photometer
dark	

Time 155	8
Time 162	7

Time	ad

```
=ad := average(2, Photometer)
```

Fig. 6. Example of a complete application. Local functions classify sensor data, pivot tables create data-constrained sheets, and collective functions aggregate data from dynamic groups.

```
if(Photometer < 20 & dark != 1) dark := 1
if(Photometer > 20 & Photometer < 70 & dark !=2)
   dark := 2
if(Photometer > 70 & dark !=3) dark := 3
```

Since these functions refer to Photometer data, each function is periodically evaluated. Depending on the latest photometer value, one of three values are assigned to the *dark* variable. Like the sensor values, Tables will keep a short history of user-defined values, enabling users to view historical *dark* values from each node. Note that even if the user stops here, this is still a useful application.

After compiling these functions, the user must define a set of data-constrained sheets using the *dark* values. This is most easily accomplished using a pivot table and selecting the *dark* variable along the sheet axis. For convenience, the user can also view other data during this process (ie: photometer values). After receiving the data from this pivot table, the user is left with three data-constrained sheets (one for each of the three *dark* values). The user can now construct collective functions that aggregate data from within these groups. In this specific instance, the collective function averages photometer values from the first group.

```
av := average(2, Photometer)
```

Since a collective function is specific to a particular *dynamic* group, the nodes participating in the collective function will change automatically. Finally, the user can create a pivot table to view the averaged photometer values. Screenshots of this application running over TelosB motes are shown in Figure 6. Although technically finished, users are free to continue experimenting with additional pivot tables and functions.

Others collective applications are similar in structure. For example, for object-tracking, the user would employ the following local functions:

```
if(Magnetometer > 20 & detection != 1)
   detection := 1 &
   wx := Magnetometer * x & wy := Magnetometer * y
if(Magnetometer < 20 & detection != 0) detection := 0
```

These functions assume that the sensor nodes have a proximity sensor and that each node stores its location in the x and y variables. When an object approaches one of the sensor nodes, the $detection$ bit is set and a weighted location is calculated. Afterwards, the user can use a pivot table to create two data-constrained sheets (for each $detection$ value).

To find the centroid of the all nodes in proximity of the object, the user can specify the following set of functions in the $detection$ 1 sheet:

```
fx := sum(3, WX) / sum(3, Magnetometer)
fy := sum(3, WY) / sum(3, Magnetometer)
```

These functions average the locations while requiring a minimum of three weighted locations. Finally, the user can construct a pivot table to view the centroids.

3 Implementation and Evaluation

Tables consists of two major components: the graphical user interface (GUI) that resides on the basestation and the runtime that resides on the sensor nodes. The GUI also executes collective functions and interacts with the sensor network via a USB-tethered mote. Having a single point of interaction with the sensor network could pose a scalability problem and we plan to experiment with a multi-master scheme in the future. Finally, the mote runtime contains communication services, and an interpreter for local functions and group management.

The Tables runtime is available for the TelosB mote [1]. The motes are equipped with a modified version of Mantis OS [3], although the Tables runtime can also be ported to other operating systems (TinyOS [12], Contiki [7]). In addition, the Tables runtime makes minimal demands on the communication subystem and can communicate over multiple protocols. Our current implementation employs a tree-based routing protocol similar to the Collection Tree Protocol [9] with link-layer acknowledgements, end-to-end retransmissions, out-of-order packet arrival, and in-network aggregation. The communication subsystem does not support flow control or packet loss, but these are areas that we expect to improve. Performance evaluation was performed over a local 25 node testbed with each node equally spaced out over approximately 20 by 15 feet. The nodes were set to the lowest radio power setting resulting in a maximum of 3 to 4 hops.

3.1 CPU

For CPU overhead, we compare the duty cycle of several applications. We caution, however, that the duty cycle in real deployments will be heavily influenced by the environment and system configuration (MAC protocols, routing, etc.). Therefore, we concentrate on the relative increases incurred by our system. Power management in Mantis

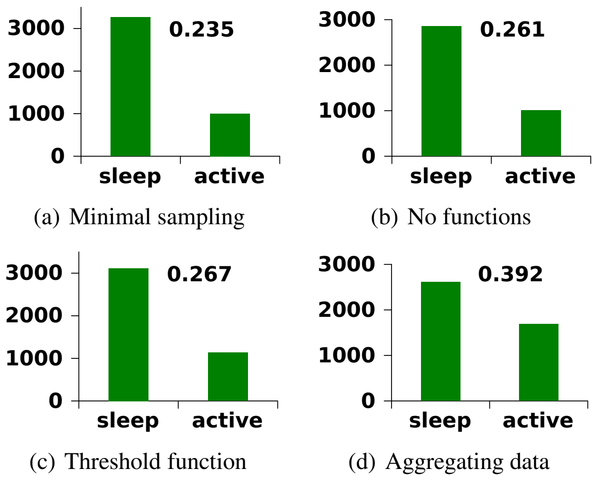


Fig. 7. Duty cycles and histograms (in timesteps) of active and sleep periods on a node

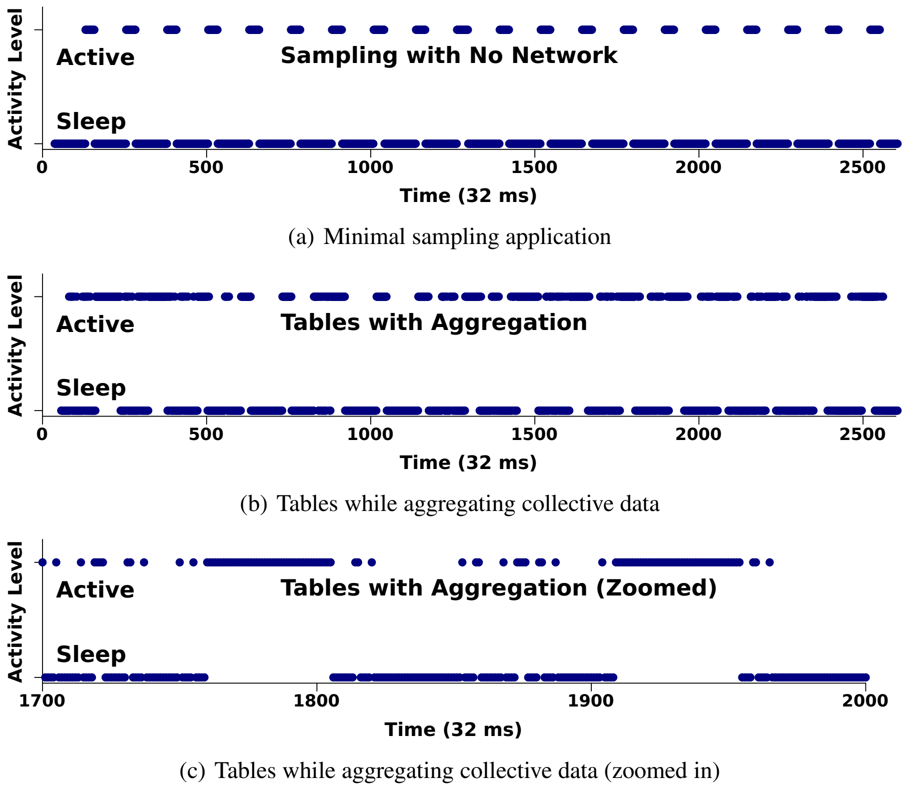


Fig. 8. Activity levels on a node over time while performing sampling (a) and aggregation (b)

OS is implicit; the node automatically enters a low-power sleep mode if there are no active threads. Threads are deactivated if the thread initiates a *sleep* command or invokes a blocking call (ie: to receive sensor or networking data). The first application we construct is a minimal sensor sampling application that runs directly on top of the operating system. This application features a single thread that sits in an infinite loop reading photometer, thermistor, and humidity data every three seconds for nine seconds. It then waits for six seconds between these sampling windows. Immediately after reading the sensor data, the values are printed over USB. As Figure 7(a) indicates, the node is active approximately 24% of the time.

A Tables application that collects and stores sensor data but does not have any functions or pivot tables, is active approximately 26% of the time (Figure 7(b)). This modest increase is due to the additional processing used to queue the sensor data, and the additional threads created by the Tables runtime. If the user includes a local filtering function for one or more of the sensors, the node becomes active 27% of the time (Figure 7(c)). Because this function refers to one of the sensor values, the function is executed whenever new sensor data is sampled. Finally, when a sensor node is performing in-network aggregation, the activity level jumps to 39% (Figure 7(d)). Although the increase is large, most applications will not perform in-network aggregation very often.

A more detailed view of the activity levels of the applications are shown in Figure 7. As visible, the minimal application exhibits a very regular structure. The node is active for a short time and then enters a low-power state. The aggregation application exhibits a less regular structure with longer active periods. Due to the resolution of the figure, some areas appear to be in both the active and sleep state. However, when the figure is zoomed in (Figure 8(c)), it is apparent there are many extremely short active states followed by periods in the lower power state.

3.2 Memory

The Tables runtime features a dynamic memory manager that initially has access to 7898 bytes (the remainder is used by the OS). The available memory is used to allocate stack space for user threads, store queued data, store local and collective functions, and to maintain group state. Tables, by default, is configured to store up to 30 sensor values per sensor queue (Photometer, Thermistor, and Humidity). After initializing the Tables runtime, the memory manager has access to 4274 bytes.

There are several sources of dynamic memory consumption in Tables. First, Tables allocates memory to store user values. Unlike a simple array of values, values in Tables are referenced by name, include a timestamp, and are used to trigger computation. Consequently, adding a data element to an existing queue uses 12 bytes. Creating a new queue uses 32 bytes. With respect to group management, the smallest group (with a single assignment function) will use 106 bytes. This is used to store group membership information, the sheet constraint, and to allocate stack space for a *publication* thread that transmits data for the collective function.

The memory used for local functions is determined by the complexity of the function. We illustrate the number of bytes consumed for a variety of assignment functions (Figure 9(a)). Simple assignments consume as little as 38 bytes (`y := 3`). Functions that reference a sensor value (`y := Photometer`) are evaluated periodically and

Function	Size (bytes)
x := 1	38
x := 1 + 2	42
x := Photometer	54
x := average(2, Photometer)	86
if(x < 3) x := x + 1	64
if(Photometer < 50) x := 0	64
if(Photometer < 50) x := average(2, Photometer)	70
if(x < min(2, Photometer)) x := 1	102

(a) Functions of varying complexity

Pivot Table	Size (bytes)
0 MD, 1 Data	50
1 MD, 1 Data	84
2 MD, 1 Data	106
0 MD, 2 Data	72
1 MD, 2 Data	106
2 MD, 2 Data	128
0 MD, 3 Data	94
1 MD, 3 Data	128
2 MD, 3 Data	150

(b) Pivot tables with varying number of metadata (MD) and data items

Fig. 9. Dynamic memory consumption of different programming components

therefore allocate space to store the function for future evaluation. For more complex assignments involving vector operations (`y := average(2, Photometer)`), the memory consumption increases to 86 bytes. Unlike other arithmetic operators, vector functions must accommodate a *minimum* history of values. A simple conditional involving arithmetic operations consumes 64 bytes and memory usage increases to 70 bytes for conditionals that assign a vector operation. Conditionals evaluating a vector function, however, consume 102 bytes. Like other vector functions, this conditional must be periodically re-evaluated and maintain a history of minimum values.

The Tables runtime must also allocate memory for pivot table processing (Figure 9(b)). Because pivot tables may request data with many elements, copying and storing all the values for the responses may easily exceed the amount of available memory. Consequently, Tables processes and transmits responses in batches to limit memory consumption. The most simple pivot table, one requesting sensor data without any metadata, consumes a total of 50 bytes. Adding a data element adds an additional 22 bytes. The first metadata element, however, adds an additional 34 bytes, while each additional metadata element adds 22 bytes.

3.3 Network

Although both pivot table processing and in-network aggregation dominate communication in Tables, publishing data for collective functions exhibits relatively low packet overhead. Because the data is aggregated in the network, only a few data values are stored in the packet. Consequently, publications for collective functions are transmitted in a single packet. Pivot tables, however, allocate more dynamic memory and must be transmitted using multiple packets. Currently, each Tables network packet has access to 88 bytes. However, after allocating space for packet processing, pivot table replies use a maximum of 74 bytes for transmission.

The number of packets used to transmit the reply will largely depend on the number of elements in the data queue. For a pivot table requesting one data item and two metadata items, and with 5 elements, it takes one packet to transmit the entire response.

Increasing the number of elements to 10 doubles the number of packets. 20 elements takes 3 packets, and 30 elements takes 4 packets to transmit the entire response. This overhead is due to the unique structure of pivot table replies. Each data element includes a timestamp and a list of metadata used to describe that data. This flexible structure is used because Tables does not differentiate system values (sensor values, ID, etc.) from user values, and users may freely mix these values in a pivot table. Also not all nodes may contain the same data and metadata elements.

4 Related Works

Tables can be compared to both typical programming models and end-user environments (Mote View [20], Microsoft SensorWeb [17], SensorBase [4]). Many features offered by end-user environments are complementary to Tables (online collaboration, user management, etc.). However unlike these environments, Tables offers a complete, integrated programming model. Typically sensor network applications are developed using low-level programming languages, such as NesC [8] to create programs with minimal overhead. Although powerful, this makes NesC a challenging programming environment even for experienced programmers using integrated development environments (Viptos [5], TOSDev [16]).

For advanced programmers, macroprogramming models offer powerful abstractions that simplify communication and tasking (Tenet [10]). Some of these models abstract neighbor information (Abstract Regions [15], Hoods [21]) or offer a single global view of the network (Kairos [11]). The EnviroSuite programming model [2] abstracts events, similar to sheet groups and users assign computation to mobile events similar to collective functions. Other functional programming models (Regiment [19]) combine declarative programming with stream processing (WaveScript [18]). Similarly, Tables can be viewed as a comprehensive macroprogramming model featuring declarative tasks, dynamic event-groups, and implicit communication with an emphasis on an iterative mode of operation.

Tables can also be compared to interactive debugging tools. Marionette [22] provides users with the ability to probe data values and dynamically invoke functions. Values from the sensor node are transmitted to the basestation, which in turn, executes the function that normally runs on the sensor node. Although Marionette adds important features to TinyOS, it does not fundamentally alter the TinyOS programming model.

5 Conclusion

We described Tables, a spreadsheet-inspired programming environment that combines data-driven functions, event-based groups, and implicit, tiered communication. We showed how to create several applications, including data monitoring and classification, with simple-to-use graphical tools. We also analyzed our prototype system for computational, memory, and networking overhead. Although our system exhibits modest overhead, we believe this overhead is acceptable for users that will benefit from our programming interface.

Tables is not necessarily ideal for all sensor network deployments. Some applications will require fine hardware control. Likewise, scenarios in which limited control is desirable, may require simple end-user interfaces. Tables sits between these two extremes. Like spreadsheets, we do not believe that Tables will or should replace low-level programming paradigms, but instead, should make sensor network programming more accessible to a larger number of people. To that end, we believe that Tables has an exciting future.

Acknowledgements

This research was partially funded by the Department of Homeland Security-sponsored Southeast Region Research Initiative (SERRI) at the Department of Energy Oak Ridge National Laboratory.

References

1. Crossbow, http://www.xbow.com
2. Abdelzaher, T., Blum, B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, S., Stankovic, J., Stoleru, R., Wood, A.: Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In: International Conference on Distributed Computing Systems (ICDCS) (2004)
3. Abrach, H., Bhatti, S., Carlson, J., Dai, H., Rose, J., Sheth, A., Shucker, B., Han, R.: Mantis: System support for multimodal networks of in-situ sensors. In: Workshop on Wireless Sensor Networks and Applications (WSNA) (2003)
4. Chang, K., Yau, N., Hansen, M., Estrin, D.: Sensorbase.org - a centralized repository to slog sensor network data. In: Euro-American Workshop on Middleware for Sensor Networks (EAWMS - DCOSS) (2006)
5. Cheong, E., Lee, E.A., Zhao, Y.: Viptos: a graphical development and simulation environment for tinyos-based wireless sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys) (2005)
6. Date, C.J.: A guide to the SQL standard. Addison-Wesley Longman Publishing Co., Inc., Boston (1986)
7. Dunkels, A., Gronvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: IEEE International Conference on Local Computer Networks (LCN) (2004)
8. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesc language: A holistic approach to networked embedded systems. In: Programming Language Design and Implementation (PLDI) (2003)
9. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: ACM Conference on Embedded Networked Sensor Systems (SenSys) (2009)
10. Gnawali, O., Greenstein, B., Jang, K.-Y., Joki, A., Paek, J., Vieira, M., Estrin, D., Govindan, R., Kohler, E.: The tenet architecture for tiered sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys) (2006)
11. Gummadi, R., Gnawali, O., Govindan, R.: Macro-programming wireless sensor networks using kairos. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 126–140. Springer, Heidelberg (2005)

12. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System Architecture Directions for Networked Sensors. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2000)
13. Horey, J., Bridges, P., Maccabe, A., Mielke, A.: Work-in-progress: The design of a spreadsheet interface. In: Information Processing in Sensor Networks, IPSN (2005)
14. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Transaction Database Systems, 122–173 (2005)
15. Mainland, G., Welsh, M.: Programming sensor networks using abstract regions. In: Symposium on Networked Systems Design and Implementation, NSDI (2004)
16. McCartney, W.P., Sridhar, N.: Tosdev: a rapid development environment for tinyos. In: ACM Conference on Embedded Networked Sensor Systems (SenSys) (2006)
17. Nath, S., Liu, J., Zhao, F.: Sensormap for wide-area sensor webs. IEEE Computer Magazine 40(7), 90–93 (2007)
18. Newton, R.R., Girod, L.D., Morrisett, J.G., Craig, M.B., Madden, S.R.: Design and evaluation of a compiler for embedded stream programs. In: ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES) (2008)
19. Newton, R.R., Morrisett, J.G., Welsh, M.: The regiment macroprogramming system. In: Information Processing in Sensor Networks (IPSN) (2007)
20. Turon, M.: Mote-view: A sensor network monitoring and management tool. In: Workshop on Embedded Networked Sensors (EmNets) (2005)
21. Whitehouse, K., Sharp, C., Brewer, E., Culler, D.: Hood: a neighborhood abstraction for sensor networks. In: International Conference on Mobile Systems, Applications and Services, MobiSys (2004)
22. Whitehouse, K., Tolle, G., taneja, J., Sharp, C., Kim, S., Jeong, J., Hui, J., Dutta, P., Culler, D.: Marionette: Using rpc for interactive development and debugging of wireless embedded networks. In: Information Processing in Sensor Networks (IPSN) (2006)
23. Woo, A., Seth, S., Olson, T., Liu, J., Zhao, F.: A spreadsheet approach to programming and managing sensor networks. In: Information Processing in Sensor Networks (IPSN) (2006)
24. Yao, Y., Gehrke, J.: The Cougar Approach to In-Network Query Processing in Sensor Networks. In: ACM SIGMOD Conference (2002)

Optimized Java Binary and Virtual Machine for Tiny Motes

Faisal Aslam[1], Luminous Fennell[1], Christian Schindelhauer[1], Peter Thiemann[1], Gidon Ernst[1], Elmar Haussmann[1], Stefan Rührup[1], and Zastash A. Uzmi[2]

[1] University of Freiburg, Germany
{aslam,fennell,schindel,thiemann,ernst,haussmann, ruehrup}@informatik.uni-freiburg.de
[2] Lahore University of Management Sciences, Pakistan
zartash@lums.edu.pk

Abstract. We have developed TakaTuka, a Java Virtual Machine optimized for tiny embedded devices such as wireless sensor motes. TakaTuka[1] requires very little memory and processing power from the host device. This has been verified by successfully running TakaTuka on four different mote platforms. The focus of this paper is TakaTuka's optimization of program memory usage. In addition, it also gives an overview of TakaTuka's linkage with TinyOS and power management. TakaTuka optimizes storage requirements for the Java classfiles as well as for the JVM interpreter, both of which are expected to be stored on the embedded devices. These optimizations are performed on the desktop computer during the linking phase, before transferring the Java binary and the corresponding JVM interpreter onto a mote and thus without burdening its memory or computation resources. We have compared TakaTuka with the Sentilla, Darjeeling and Squawk JVMs.

1 Introduction

A common way of programming an application for wireless sensor motes is by using a low level programming language such as Assembly, C and NesC [3]. These languages tend to have a steep learning curve and the resulting programs are difficult to debug and maintain. In contrast, it is attractive to program these motes in Java, a widely used high level programming language with a large developer community. Java is highly portable and provides many high level concepts including object oriented design, type safety, exception handling and runtime garbage collection. However, Java portability requires a virtual machine, which comes with significant memory and computation overhead [7]. Therefore, it is difficult to run such a virtual machine on a wireless sensor mote which typically has a 16 or an 8 bit microcontroller with around 10KB of RAM and 100KB of flash [18]. Furthermore, these motes cannot perform computation intensive tasks due their limited battery lifetime. Therefore, a Java Virtual Machine (JVM) designed for such tiny embedded devices must have low computation requirements,

[1] The complete source of TakaTuka is available athttp://takatuka.sourceforge.net

a small RAM footprint, and small storage requirements. To satisfy these stringent requirements, we have designed TakaTuka, a JVM for wireless sensor motes.

The focus of this paper is TakaTuka's optimization for reduction in flash storage requirements of the Java classfiles and virtual machine. The main contributions described in this paper are: 1) Extensive Java bytecode and constant pool optimizations. As part of these, we present a novel *optimal* bytecode replacement algorithm (Section 3.3.2). 2) A condensed Java binary format (called Tuk). Beside providing size reduction, the Tuk file format reduces RAM and computation requirements by enabling constant time access to preloaded information stored in flash. 3) A novel design for dynamically sizing the JVM interpreter. 4) TakaTuka Java interfaces for typical mote hardware and an implementation based on TinyOS. The current version of TakaTuka can supports *all* mote platforms using TinyOS that either have MSP430 or AVR family processors. We have acquired four of those platforms and successfully tested TakaTuka on them. This includes Crossbow's Mica2, Micaz, TelosB and Sentilla's JCreate [18] [17].

Outline: We present related work and background information in Section 2. Java bytecode optimization is described in Section 3 and constant pool optimization in Section 4. The Tuk file format, is discussed in, Section 5 and our JVM design is presented in Section 6. We discuss TakaTuka's linkage with TinyOS in Section 7. We present results in Section 8 and finally, in Section 9, we draw the conclusions.

2 Related Work and Background

This section summarizes the existing JVMs for tiny embedded devices and bytecode size reduction techniques.

JVM for Motes: Recently, Sun Microsystems has developed *Squawk*, a JVM for embedded systems [7], which overcomes some of the shortcomings of traditional JVMs by employing a *Split VM Architecture* (SVA). In SVA, resource-hungry tasks of the JVM, including class file loading and bytecode verification are performed on the desktop [14] [7]. This process reduces the memory and CPU usage requirements for the execution of the program on the mote, because no runtime loading and verification is required. When compared to standard JVMs, Squawk has less stringent requirements for resources; it is, however, still not feasible to run Squawk on a typical mote equipped with an 8-bit microcontroller, a few hundred KB of flash and around 10KB of RAM [7]. For these typical motes, Sentilla Corp. has developed a JVM, but it is not open-source and currently does not support any devices other than Sentilla motes [17]. Darjeeling, is an open source JVM designed for motes [11]. It does support a good part of the JVM specification but sacrifices some features like floating point support, 8-byte data types and synchronized static-method calls for efficiency [11]. There are a few other JVMs available for embedded devices, such as NanoVM [1], and VM* [10], but these are either limited in functionality by not fully supporting JVM specifications or are closed source with a limited scope of operation only on specific devices. TakaTuka aims to remain small, open source and suitable for a wide variety of embedded devices while providing all features of a *fully CLDC-compliant JVM*[2]. The current version of TakaTuka supports all but two of the Java bytecode instructions and most of the CLDC library. We also support threading, synchronized method calls, 8-byte data types and 4-byte floating point arithmetic on motes.

Bytecode Optimization: The two primary methods for bytecode size reduction are *compression* and *compaction* [6][5]. A typical compression technique requires partial or full decompression at runtime [6]. Any decompression always results in computation and memory overhead. Therefore, performing decompression at runtime is not desirable for embedded devices. In contrast to compression, *compaction* involves identifying and factoring out recurring instruction sequences and creating new customized bytecode instructions to replace those sequences [2]. These customized instructions do not require decompression and are interpretable because they share the characteristics of normal bytecode instructions. The process of compaction produces a smaller code that is executable or interpretable with no or little overhead. The compaction scheme given in [2] is shown to produce bytecode that is about 15% smaller and runs 2 to 30% slower than the original bytecode. Rayside et al. [5] compaction scheme produces bytecode that is about 20% smaller. In contrast to above mentioned compaction approaches, TakaTuka comprehensive compaction scheme produces a bytecode reduction of about 57% on average and the resultant bytecode runs faster without using any extra RAM.

3 TakaTuka Bytecode Compaction

We employ three bytecode compaction techniques, each of which replaces a single bytecode instruction or a sequence of bytecode instructions with a new *customized bytecode instruction* such that the total size of the bytecode is reduced. A customized bytecode instruction, like any other bytecode instruction, is composed of an opcode and an optional set of operands. In the following, we first explain the process of choosing an opcode for a customized instruction. Then, we provide the details of the compaction processes and relevant algorithms used in TakaTuka.

3.1 Available Opcodes

Each customized instruction uses an opcode that is not used by any other bytecode instruction. Hence the cardinality of the set of available opcodes impacts the extent of compaction. The Java specification has reserved one byte to represent 256 possible Java opcodes but uses only 204 of those for corresponding bytecode instructions [14]. Thus, there are 52 unused opcodes that are available for defining customized instructions. Furthermore, the Java specification includes many bytecode instructions with similar functionalities but different data-type information. The type information of such instructions is only used

[2] We might never actually obtain formal CLDC-compliance due to the high price tag associated with the license of the CLDC Technology Compatibility Kit (CLDC TCK).

in Java bytecode verification and is not required by the JVM interpreter. Since the Split VM architecture (SVA) does not require run-time verification (see Section 2) additional 29 opcodes for a compaction algorithm are available after completing bytecode verification during linking phase. Finally, many Java programs may not use all the 204 standard Java instructions, depending upon the functionality of the program. Hence a custom-made JVM interpreter such as the one offered by TakaTuka can make use of additional opcodes, not used by the Java program, for the purpose of defining customized instructions during the bytecode compaction process.

3.2 Single Instruction Compaction

In this technique of compaction, size of a single bytecode instruction is reduced by replacing it with a smaller customized instruction. That is, in single instruction compaction, each customized instruction replaces only a single bytecode instruction. The single instruction compaction in TakaTuka can either be a reduction in the memory footprint needed to represent an operand (called *Operand reduction*) or a complete removal of the Operand from the bytecode instruction (called *Operand removal*).

Operand reduction: Many instructions in standard Java use either a 2-byte constant pool (CP) index or a 2-byte operand as a branch offset [14]. In TakaTuka, we introduce a new custom instruction with a reduced operand size of one byte, if the operand value is smaller than 256. In order to maximize the savings resulting from the use of this technique, we sort the information in our set of global CPs[3] such that most referred entries of a CP from the bytecode are stored at a numerically small CP index. This leads to a large number of reduced size constant pool instructions in the bytecode.

Operand removal: In TakaTuka, we also combine the opcode and operand to form a customized instruction with implicit operand(s). For example, the instruction `ILOAD 0x0010` could be converted to `ILOAD_0x0010` and the two bytes originally used by the operand may be saved. Note, however, that we do not apply operand removal on offset-instructions as their offset usually changes after any kind of bytecode compaction.

3.3 Multiple Instruction Compaction (MIC)

In multiple instruction compaction (MIC), a recurring sequence of instructions, called a *pattern*, is replaced by a single *customized instruction*. For example, a pattern {`GOTO 0x0030, LDC 0x01, POP`} in bytecode could be replaced by a single customized instruction `GOTO_LDC_POP 0x003001`, providing a reduction of two bytes per occurrence. Note that, the MIC technique perform compaction in the opcodes only, without affecting the storage needed for operands. The MIC

[3] Global constant pool is explained in detail in Section 4.

technique involves finding a set of valid patterns and then replacing a subset of those patterns with customized instructions. First, we define the criteria that must be met for a pattern to be replaceable and then we discuss the valid pattern search and replacement algorithms used in TakaTuka.

Valid Pattern Criterion: A sequence of bytecode instructions or a pattern is said to be *valid* if it can be replaced by a single customized instruction. A valid pattern fulfils the following two criterion: 1) A branch-target instruction can only be the first instruction of a pattern, and 2) Any Java bytecode instruction designed to invoke a method can only be a last instruction of a pattern. Note that above restrictions are imposed to avoid extra computation or RAM required for decoding a customized instruction during runtime for finding a return or branch offset target inside it.

3.3.1 Pattern Identification

The pattern identification algorithm finds and selects a number of patterns of instruction sequences from the original bytecode of the Java program, up to a maximum number of available opcodes. These patterns are stored in a hash-map which is used as an input to the pattern replacement algorithm. The pattern replacement algorithm then constructs customized instructions and replaces the input patterns with those customized instructions in the bytecode. We use the following terminology to explain the pattern identification and replacement algorithms.

m :	Total number of opcodes that may be used by a customized instruction.
k :	Maximum number of single instructions in any pattern.
l_i :	Number of single instructions in a pattern i that can potentially be replaced by a customized instruction. We also refer to this parameter as the length of the pattern i.
ϵ_i :	Reduction in bytecode achieved when one occurrence of a pattern i in the bytecode is replaced by a customized instruction. ϵ_i equals $l_i - 1$ when a pattern i is replaced by a MIC customized instruction.
ζ_i :	Frequency of a pattern i, that is to be replaced by a customized instruction, in the entire bytecode of the Java program.
η_i :	Total reduction (i.e. $\epsilon_i \cdot \zeta_i$) in bytecode achieved when a pattern i is replaced by a customized instruction, in the entire bytecode of the Java program.
$\xi_{(y)}$:	$\sum\limits_{i \in y} \eta_i$ where y is a set of patterns.

In TakaTuka, pattern generation for multiple instruction compaction uses a Multi-pass greedy algorithm, which is based on a simple Single-pass greedy algorithm.

Single-pass greedy algorithm: The Single-pass greedy algorithm creates a list of patterns of length $\leq k$ by traversing the bytecode exactly once. When a valid pattern i of any length is encountered the first time, it is added to the hash map with $\zeta_i = 1$. Then, ζ_i is incremented whenever the same pattern i

is found again while traversing the remaining bytecode. Consequently, after a single traversal of the Java bytecode, the hash map contains all possible patterns of length $\leq k$ with their corresponding frequencies. The algorithm returns a subset σ of patterns from within the hash map such that $|\sigma| \leq m$ and $\xi_{(\sigma)}$ is maximized. This algorithm has one major flaw: it returns many patterns that are not new pattern but subset of other longer patterns, undermining the extent of bytecode reduction.

Multi-pass greedy algorithm: The multi-pass greedy algorithm mitigates the limitation described above by traversing the bytecode multiple times, making temporary changes in each iteration, and using that changed bytecode in subsequent iterations. In the first iteration, the single-pass greedy algorithm is used on a copy of the bytecode and the resulting patterns are stored in a set y. A pattern i is then selected, such that $\eta_i \geq \eta_j \; \forall j \in y$, and replaced as a customized instruction in the copy of the bytecode. Subsequent iterations are similar except that the single-pass greedy algorithm is called on the modified copy of bytecode from the previous iteration. This continues until either m patterns of length $\leq k$ are selected or additional patterns cannot be found. The customized instructions introduced in a given iteration may become a part of a new pattern in subsequent iterations, as long as the constraint of maximum k original single instructions per customized instruction is not violated.

3.3.2 Pattern Replacement

The pattern replacement algorithm takes the bytecode and a set of patterns and replaces those patterns, as they appear in the byteode, by new customized instructions. The primary goal in this replacement process is to maximize the bytecode reduction, leading to the maximum savings in storage. While our pattern generation algorithm is greedy and may not generate an optimal set of patterns, our pattern replacement algorithm is not only *optimal* but also runs in polynomial time. First, we describe our algorithm, then we show its polynomial complexity in Theorem 1, finally proving its optimality in Theorem 2.

Algorithm: The pattern replacement algorithm keeps track of many temporary solutions in order to produce the replacement with the maximum savings. The algorithm is applied on each class method within the bytecode one by one and produces the maximum reduction possible for that method with the given set of pattern. The inputs to the pattern replacement algorithm are: 1) the number k indicating the maximum number of single instructions in any pattern, 2) a set of patterns σ generated by a pattern identification algorithm and 3) the bytecode of a method. The pattern replacement algorithm creates a tree with different replacement possibilities. One branch of this tree contains the bytecode sequence corresponding to the maximum reduction in bytecode, and the algorithm uses this branch to update the bytecode of the method. To demonstrate the replacement algorithm, assume that the instruction at index i in the bytecode of a method μ is represented by τ_i. That is, $\{\tau_1, \tau_2, ..., \tau_\lambda\}$ represents the method bytecode, where λ is the total number of instructions in μ. Each level in the tree corresponds to the index in the bytecode, hence the tree has depth

λ. Within the tree, each node located at level j corresponds to either τ_j, or to a customized instruction that ends at τ_j. Each node x in the tree has exactly one incoming edge whose weight $w(x)$ is given by:

$$w(x) = \begin{cases} 1 & x \text{ is the root node} \\ 0 & x \text{ corresponds to a} \\ & \text{customized instruction,} \\ \min\left(k-1, w(x_p)+1\right) & x \text{ corresponds to } \tau_j \text{ itself} \end{cases} \quad (1)$$

where x_p is the immediate parent of node x. Note that each node x, other than root node, has exactly one immediate parent node. Each node x in the tree has at most $w(x)+1$ child nodes each corresponding to instructions with unique lengths ranging from 1 to $w(x)+1$. If node x exists at a level j in the tree, then one of its child nodes corresponds to the instruction τ_{j+1} which has length 1; each of the other $w(x)$ child nodes corresponds to a customized instruction c that represents a pattern obtained by traversing node x and parents further above in the tree, such that $2 \leq l_c \leq w(x)+1$. Each node of the tree maintains the total bytecode reduction achieved by the tree branch ending at it.

The tree is built level by level, where addition of each level is done in two phases: the *creation phase* and the *pruning phase*. The creation phase of a level j is carried out simply by finding the children of all the nodes at level $j-1$. In the pruning phase of level j, first we prune all those nodes from level j which represent an invalid customized instruction i.e. its corresponding pattern is not a member of σ. Subsequently, additional nodes are pruned such that no two nodes have the same weight on their incoming edge: if multiple nodes have w as the weight on their incoming edge, the one corresponding to the leaf of the branch with highest total bytecode reduction is kept and the remaining nodes are pruned. In this additional pruning, random selection is made if there is a tie.

Each level of the final tree has at most k nodes because the weight allowed on any incoming edge is between 0 and k-1 (see Equation 1) and after pruning nodes on each level have edges with distinct weights. Thus, the resulting tree is a linear sized structure with depth λ and a constant span k. After the tree with level λ is completely constructed, the leaf node with the highest saving is identified and the corresponding tree branch is used to replace the bytecode.

Theorem 1. ***The time complexity of the replacement algorithm is*** *$O(k^2 \cdot \lambda)$.*

Proof. The complexity of the algorithm depends on the size of the tree and the number of operations performed on each node. From the description of the algorithm, we note that the depth of the tree is exactly λ and each level of the tree has at most k nodes. The final tree has at most $k \cdot \lambda$ nodes. However, in the worst case, before a pruning phase, a level j may have a total of up to $1+2+3+...+k$ nodes. Pruning nodes at level j means finding the node with the maximum reduction for each weight $0 \leq w \leq k-1$. This can be done with an effort proportional to the number of nodes at this level, i.e. $\frac{1}{2}k(k+1)$ nodes in the worst case, which is bounded by $O(k^2)$. Since there are exactly λ levels in the tree, the worst case complexity of the algorithm is $O(k^2 \cdot \lambda)$.

Lemma 1. ***All nodes with same weight w on their edge at level j will have same sub-tree originating from them.***

Proof. We prove our claim by contradiction: assume that all nodes with the same weight w on their incoming edges at level j do not have the same sub-tree originating from them. This is possible if and only if their immediate children are different. We now consider two situations: (i) when w is zero, each node has only one child node corresponding to the instruction τ_{j+1}. Thus, all sub-trees are the same, contradicting the original assumption, and proving the lemma. (ii) when w is non-zero, a child node either has a customized instruction composed of at most w parent nodes or a simple instruction τ_{j+1}. Based on Equation (1), all of those w parents of each node will always be simple instructions instead of customized instructions. The algorithm says that a simple instruction τ_i can only occur at level i in tree. Hence all of those w parents nodes must be equal. In summary, the immediate children of each node with same w at level j will have the same w parents and, therefore, will have the same child nodes. Hence the sub-tree emanating from each node with same weight w at level j will always be the same.

Theorem 2. ***Given a set of patterns, the pattern replacement algorithm finds the replacement with maximum overall reduction in the size of bytecode.***

Proof. First, we note that the complete tree without pruning contains all combinations of solutions including the optimal one identified by the leaf node with the highest total reduction. Next, we argue that the branch corresponding to the optimal solution is not affected by pruning. Using Lemma 1, all nodes with the same weight w on their incoming edge at level j have the same sub-tree emanating from them. Therefore, pruning all nodes with the same weight w except the one with maximum saving achieved so far, implies that the optimal solution is still part of the tree.

4 TakaTuka Constant Pool Optimization

Each class file has a collection of distinct constant values of variable size called the *Constant Pool* (CP). These constant values either define the characteristics of the class or are used by the class itself. A two byte index is used to access a given CP value which is usually larger than two bytes and is used multiple times from the class file. The aggregated CP size of a project is usually much larger compared to its total bytecode size. Hence, reducing CP size is critical in the overall size reduction of a Java program. Our constant pool design is based on some of the ideas given in [5] and [13] with improvements drawn using the characteristics of a Split-VM architecture, which is not considered in above references. In the following sub-sections we present the optimizations we used in TakaTuka for reducing the CP size.

Global Constant Pool: Each value of a CP entry could be of one of the eleven different types as specified in the Java specification [14]. In traditional designs, the CP values of a single class appear in an arbitrary order within the CP, where a leading one byte tag is used for type identification. This design, however, has the following shortcomings: 1) One byte is consumed to specify the type with each CP entry. 2) Since a CP is unordered, an index has to be built in RAM in order to index its entities in a constant-time. 3) Although CP values are distinct for one class, there can be many redundant occurrences in the scope of a given project.

The above mentioned shortcomings lead to excessive flash and RAM requirement, both of which are scarce resources in sensor motes. To address this in TakaTuka, we use the preloading characteristic of SVA and create one *global* pool per type, during the linking phase. As compared to traditional CPs, our set of Global Constant Pools (GCPs) have no redundant information per project. We keep a common header for these GCPs specifying the start address of a pool and corresponding type. As all entries of a GCP have the same type, no tag is required per constant pool entry. Keeping a separate CP per type enables a constant time lookup for any CP's entry in the flash and does not require loading the complete CP in RAM. This is because each CP type has only fixed size values[4] hence given a constant pool type one can directly jump to a specify CP index by computing the offset from the first entry of the same type. This constant time lookup is possible because each Java CP-instruction always accesses the same type of CP (e.g., `INVOKEVIRTUAL` always access CP of type 10). However, there are three exceptions to this namely the instructions `LDC`, `LDC_W` and `LDC2_W`. We have introduced five additional bytecode instructions so that in TakaTuka each CP instruction, including the ones mentioned above, implicitly contain the CP type information.

Reference Resolution: Traditional JVMs apply dynamic loading, also called on-demand loading. Whenever a class method or a field needs to be accessed, the corresponding class file has to be loaded into RAM after performing verification, preparation and resolution [14]. To resolve references during runtime, fully qualified names are required to identify components (i.e. methods, fields and, classes). In TakaTuka, we have used the preloading characteristic of SVA to resolve names during linking. Hence a preloaded, preverified and resolved Java program is transferred to a mote. This allows us to remove all the UTF-8 strings traditionally required for name resolution but not used by the application. Furthermore, we can also remove all the other constant pool entries (e.g., all entries of type 12) typically used for resolving names during runtime[14].

[4] UTF-8 constant pool type has variable length values but they are never directly used from within the bytecode [14].

5 Tuk File Format

The classfile has three important parts: the bytecode, the CP and the structure information. We observed that the structure information usually makes up for half the size of classfiles of a program. We use a special format called *Tuk* for storing the set of classfiles used by a user-program and corresponding Java library components. The Tuk format has two main characteristics. First, it only contains information required during program execution in a reduced format, forgoing any information required during linking phases. Second, the Tuk format contains preloaded information stored in an easy to access manner, obviating the need to load the Tuk file in a mote's RAM. This implies that different portions of a Tuk file can be accessed in constant time using pre-calculated addresses, relieving the computation resources of the host device. It may be noted that the addresses and indexes that make up part of the preloaded information are created by processing on a desktop computer, before transferring a small preloaded and preverified Tuk file to the host device. For example, in a Tuk file a CP entry for a class or a method reference contains the address of the location inside the Tuk file where actual data (i.e., the classfile or the method) resides. All the addresses in a Tuk file are relative to its starting address and are therefore platform independent. We use either 2-byte or 4-byte addresses depending upon the total size of the Tuk file.

6 TakaTuka JVM Design

In this section, we present the design of TakaTuka's dynamically customizable JVM interpreter, and its bytecode compaction support.

Customized JVM: The flash memory in some tiny embedded devices may be too small to contain a complete JVM. To address this limitation, the default behavior of TakaTuka is to reduce the size of the interpreter depending upon the set of bytecode instructions used by a given Java program. To this end, TakaTuka removes all unused components from the JVM, stripping the JVM bytecode support down to the bytecode set used by the given program. For a given Java program, the initial step of compilation is to generate a *header file* enlisting the set of bytecode instructions used by that program. This step is completed on the desktop computer. Subsequently, the JVM recompiles to shrink itself based on the information contained in the header file. This default behavior leads to a very small JVM interpreter, albeit the one that is capable of running only a subset of Java programs. If a more generic interpreter, capable of supporting additional bytecode instructions, is needed, TakaTuka allows this through a configuration file. For example, it is possible to have a JVM interpreter version that supports the complete set of bytecode instructions except the ones which involve floating point operations. Similarly, a user can also completely turn off the JVM customization, resulting in the generation of a general purpose JVM that supports any CLDC-compliant Java program.

Bytecode Compaction Support: The TakaTuka interpreter is implemented by using the *labels-as-values* approach [4] to provide direct threading. Each bytecode instruction translates into a source code snippet following a corresponding *label*. To this end, TakaTuka treats the customized instructions, resulting from single instruction compaction and multiple instruction compaction, like any other bytecode instruction that could not be compacted. However, because some instructions are only generated during the compaction process a 'static' set of labels is not applicable. TakaTuka addresses this problem by dynamically generating the labels supporting the required set of bytecode instructions. In case a user wishes to generate a general purpose JVM interpreter that can run any CLDC-compliant program then a set of *fixed* customized instructions is used, instead of generating new customized instruction per application.

7 TakaTuka Linkage with TinyOS

TakaTuka can run on all devices currently supported by TinyOS. Furthermore, TakaTuka provides a Java interface for each driver and a user can access the class implementing such an interface using only a factory-method [8], making the actual implementation hidden from the users of the driver. Therefore, TakaTuka can run on any platform that provides an implementation of those Java interfaces using any operating system, including but not limited to TinyOS [5].

To work with TinyOS, the TakaTuka interpreter is allowed to execute n bytecode instructions in a scheduling cycle before returning the control back to the TinyOS scheduler. This is accomplished by posting a TinyOS Task [12] that calls the TakaTuka interpreter which is written in C. We keep n small so that TinyOS remains responsive in between TakaTuka scheduling cycles. TinyOS is an event driven operating system. In contrast Java language functions are generally blocking and it is up to Java user to create threads when multitasking is required. Hence when a TakaTuka user calls a method that is written as event in TinyOS then that method blocks until the corresponding event is generated in TinyOS. When TinyOS receives an event, then the Java thread waiting for that event is notified. A TakaTuka user can access any of the TinyOS Commands [12] using TakaTuka native method support. For a Split-phase TinyOS Command [12], the current thread is put into a waiting state after calling the Command and is notified when latter is completed. TakaTuka also supports thread synchronization, that is used for sharing resources (e.g. radio) between multiple user threads.

Power Management: The current version of the TakaTuka JVM uses the same power management strategies as implemented in TinyOS but in future versions, we intend to provide a more comprehensive power management. In the current version, when all Java threads are sleeping or waiting for events then TinyOS could go to the low power state and TakaTuka follows through. In case an event arrives, then any thread waiting for that event is notified and and TakaTuka resumes. Furthermore, TakaTuka uses TinyOS radio drivers, thereby supporting the low-power listening implemented by TinyOS.

[5] The current version of TakaTuka has also limited support for Contiki OS.

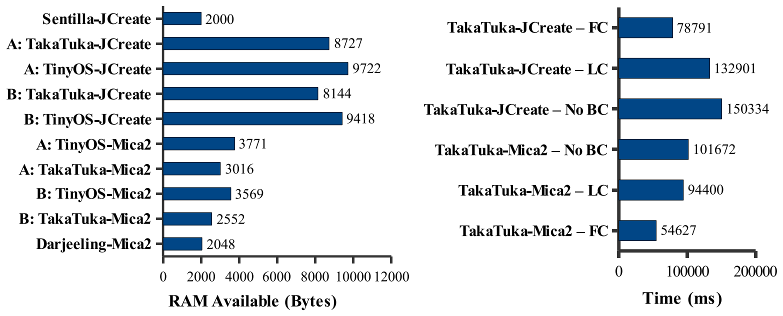


Fig. 1. Left-Fig: RAM available (in bytes) for a user program. Unlike the Sentilla and Darjeeling JVMs, the RAM available for a user program in TakaTuka and TinyOS depends on drivers used by it. We have presented two kinds of results for TinyOS and TakaTuka. a) When only serial drivers are used and b) When serial, radio and LED drivers are used. **Right-Fig:** Time (in ms) required to sort an array of 1500 bytes using the Quicksort algorithm. TakaTuka results are shown when no Bytecode Compaction (no BC), Full Compaction (FC) and Limited Compaction (LC) is used.

8 Discussion and Results

This section presents TakaTuka storage results as compared to the Darjeeling, Sentilla and Squawk JVMs.

8.1 Execution Speed and RAM

We show that our storage optimization does not have an adverse effect on RAM usage and performance. In contrast the bytecode optimization results in enhanced performance.

RAM available for a user program: The TakaTuka interpreter and the low level drivers require a few hundred bytes of statically allocated data memory. The exact amount depends on the used functionality. Furthermore we assume a worst case upper bound of 500 bytes for the stack memory required by the interpreter's routines. The remaining RAM is available as Java heap and stack memory for the user program and the runtime library. It is shown in Fig. 1, that TakaTuka has 81.44%, TinyOS has 94.18% and Sentilla JVM has 20% of the total RAM available for a program that uses serial, radio and LED drivers on a JCreate mote. For the Mica2 mote Takatuka has 62.30%, TinyOS has 87.13%, and Darjeeling has 50% of the total RAM available for the user program when LED, radio and serial drivers are used. The Darjeeling JVM does not adjust the amount of RAM available automatically. It can be set at compile time and defaults to 2048 bytes for Mica2.

Execution speed: The increase of execution speed when using bytecode compaction is shown in Fig. 1. On JCreate, the TakaTuka JVM runs 47.59% faster with Full Compaction and 11.6% faster with Limited Compaction as compared to when no compaction is used. Bytecode compaction results in a similar speed increase on Mica2.

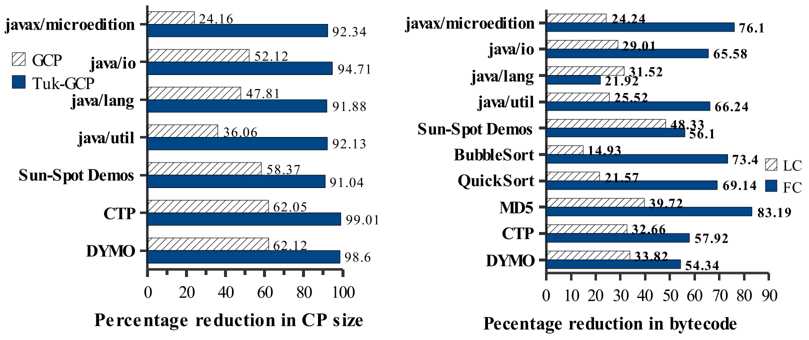


Fig. 2. Left-Fig. Percentage reduction in GCP and Tuk-GCP sizes as compared to original CP sizes of classfiles. **Right-Fig.** Percentage reduction in bytecode using full compaction (FC) and limited compaction (LC).

8.2 Storage

In this section we present TakaTuka storage optimization results.

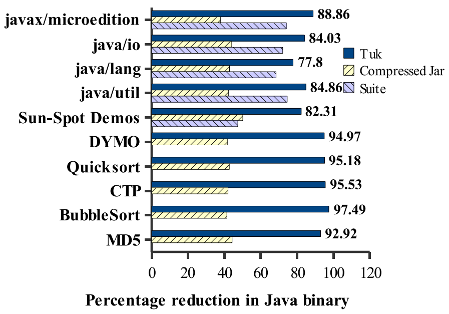
Input applications: In addition to simpler well known algorithms (Quicksort, Bubblesort, MD5) we have used DYMO and collection tree (CTP) routing protocols as input to our optimization algorithms[9][16]. Both Dymo and CTP are completely written in Java using multiple threads. Our input files to the TakaTuka optimization framework for above mentioned applications include user defined classfiles as well as all other linked library files used by the user program. We have also used CLDC library packages and Sun-Spot Demos to produce storage results. However, in this case, the input files do not include external packages and no dead-code removal is applied.

CP optimizations: We first create a set of 11 Global Constant Pools (GCPs) with no redundant information for each application, subsequently the reference resolution information is removed (Tuk-GCPs). On average the aggregated size of GCPs are 48.96% and the Tuk-GCPs are 94.24% smaller then the original corresponding classfiles aggregated CPs (Fig. 2).

BC optimizations: Our bytecode optimization not only reduces the overall Java binary size but also increases execution speed, as shown in the previous Section 8.1. In case a customized TakaTuka JVM is used, the bytecode compaction results in regenerating the interpreter's labels for new customized instructions (Section 6). Hence we define the net reduction in the Java binary size due to bytecode compaction for a given pattern x by:

$$R(x) = \eta_x - \gamma_x \tag{2}$$

where η is defined in Section 3.3.1 and used here in the context of all three compaction techniques. γ_x is the total change (increase or decrease) in the size of the label's block for supporting a customized instruction that is replacing a given pattern x of original Java instructions. Through configuration, we allow two compaction strategies in TakaTuka: Limited Compaction (LC) and Full



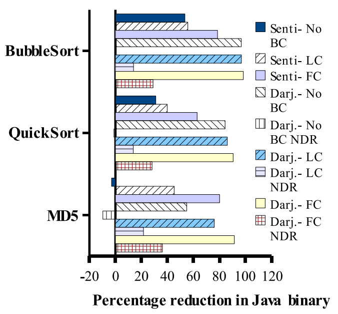
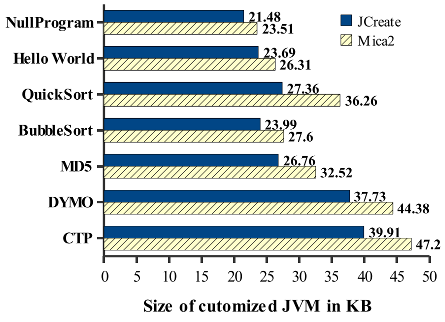


Fig. 3. Left-Fig. Overall reduction in Java binary for TakaTuka applications, Sun-Spot demos and CLDC libraries when LC is used. The applications include user and corresponding library code. In contrast, the Sun-Spot demos as well as the CLDC libraries do not include classfiles from external packages. **Right-Fig.** Overall reduction in Java binary for TakaTuka compared to the Sentilla (Senti) and Darjeeling (Darj) JVMs. The different configurations for TakaTuka are: without bytecode compaction (No BC), with Limited Compaction (LC), with Full Compaction (FC), and without dead-code removal (NDR).



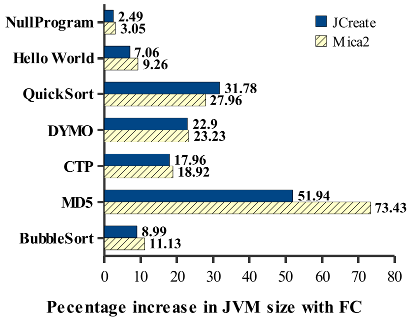


Fig. 4. Left-Fig. Overall size (in bytes) of the customized TakaTuka JVM for different applications. **Right-Fig.** The percentage increase in customized TakaTuka JVM size when Full Compaction (FC) is used.

Compaction (FC). In LC the change in size of the label's block is taken into account and compaction is only performed when the compaction reduction function of Equation 2 returns a positive value. In contrast, in FC any change in the size of the label's block is ignored and compaction is always performed if unused opcodes are available. As shown in Fig. 2, on average FC reduces the size of the Java bytecode by 62.39% and LC by 30.14%. By default we use LC instead of FC on JCreate as the net size of the Java binary and the JVM interpreter is increased when FC is used as shown in Fig 4. On Mica2 we can use FC to profit from the performance increase as its bigger flash memory can tolerate the storage increase more easily.

Overall Java binary reduction: It is shown in Fig. 3, that for the given programs, TakaTuka's Java binary file, called Tuk, is on average 24.12% smaller compared to corresponding Suite files. Furthermore the Tuk file size is reduced by 95.22% for applications and 83.57% for libraries compared to corresponding uncompressed Jar files. The right part of Fig. 3 compares the size of Tuk files resulting from different compaction configurations with the Java binaries produced by Sentilla and Darjeeling. In the comparisons with Sentilla only the user application is considered, as the library is already present on the Sentilla motes and generally not recompiled. In the comparisons with Darjeeling we also compare the size of the Tuk file without dead-code removal, as the Darjeeling JVM do not perform dead-code removal. TakaTuka always shows significant reduction in the size of the Java binary when using bytecode compaction.

Customized JVM reduction: As discussed in Section 6, TakaTuka supports both customized JVM per user program as well as general purpose JVM that can run any CLDC program. The overall size of the general purpose JVM for Mica2 is 56914 bytes and 34538 bytes for JCreate mote. Hence a customized JVM for the `Null Program` is 58.70% and 37.80% smaller compared to the general purpose JVM on a Mica2 and JCreate respectively, as shown in Fig. 4. The figure also shows customized JVM sizes for other programs. Darjeeling produces a general purpose JVM of 60800 bytes.

9 Conclusion

We have developed TakaTuka, a JVM that runs on platforms with RAM as small as 4KB and flash as small as 48KB. Using TakaTuka, we also developed a number of Java WSN applications and verified that their storage requirements are kept small on tiny platforms without increasing CPU and RAM usage. TakaTuka takes several measures to curtail the storage requirements, that include: 1) A condensed format for classfiles called Tuk. 2) Comprehensive CP reduction techniques. 3) Various bytecode reduction strategies including single and multiple bytecode compaction and a novel optimal bytecode replacement algorithm. 4) A JVM that may be customized for a user application.

These measures have resulted in an average size reduction of 95.22% for the resulting Tuk files, given a set of user Java applications, as compared to the original classfiles. Furthermore, the customized TakaTuka JVM size is shown to reduce up to 58.70%.

TakaTuka bytecode compaction results in a performance gain of 47.59% on average. This is because the customized instructions reduce the overall time needed for instruction dispatch and fetching instruction operands, which are the primary time-consuming tasks of typical JVM interpreters [15]. Furthermore, each entry in a global CP or any other part of Tuk file is accessable in a constant time, leading to reduced computation and RAM requirements at run-time.

In summary, our results for a variety of applications on a multitude of hardware platforms indicate that TakaTuka is a very promising platform for WSN applications.

References

[1] AREXX engineering. The NanoVM - Java for the AVR, http://www.harbaum.org
[2] Clausen, et al.: Java bytecode compression for low-end embedded systems. ACM Trans. Program. Lang. Syst. (2000)
[3] Gay, D., et al.: The nesc language: A holistic approach to networked embedded systems. In: ACM SIGPLAN PLDI (2003)
[4] Gregg, D., et al.: A fast java interpreter. In: The Workshop on Java (2001)
[5] Rayside, D., et al.: Compact java binaries for embedded systems. In: CASCON (1999)
[6] Saougkos, D., et al.: Revisiting java bytecode compression for embedded and mobile computing environments. IEEE Trans. Softw. Eng. (2007)
[7] Simon, D., et al.: Java on the bare metal of wireless sensor devices: the squawk java virtual machine. In: ACM SIGPLAN VEE (2006)
[8] Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1994)
[9] Chakeres, I., et al.: Dynamic MANET On-demand (DYMO) Routing. IETF (2008)
[10] Koshy, J., et al.: Vmstar: Sythesizing scalable runtime environments for sensor networks. In: SenSys (2005)
[11] Brouwers, N., et al.: Darjeeling, a feature-rich vm for the resource poor. In: SenSys (2009)
[12] Levis, P., et al.: TinyOS Programming. Cambridge University Press, Cambridge (2009)
[13] De Wang, S., et al.: Jato: A compact binary file format for java class. In: ICPADS (2001)
[14] Lindholm, T., et al.: The Java Virtual Machine Specification. Prentice-Hall, Englewood Cliffs (1999)
[15] Shi, Y., et al.: Virtual machine showdown: Stack versus registers. In: VEE (2005)
[16] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Culler, D.E., Liu, J., Welsh, M. (eds.) SenSys, pp. 1–14. ACM, New York (2009)
[17] Sentilla, http://www.sentilla.com
[18] Crossbow Technology. Wireless Sensor Networks, http://www.xbow.com

ZeroCal: Automatic MAC Protocol Calibration

Andreas Meier, Matthias Woehrle, Marco Zimmerling, and Lothar Thiele

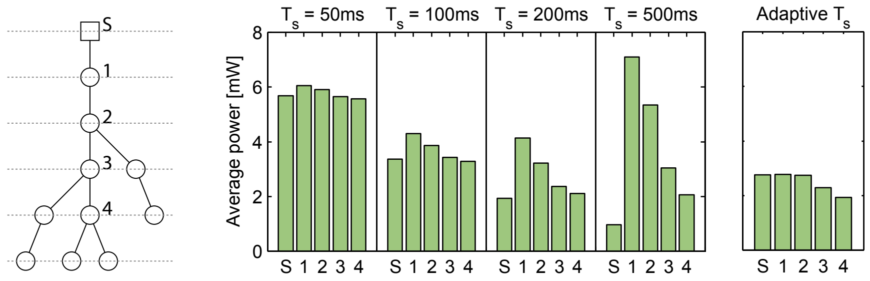
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
{a.meier,woehrle,zimmerling,thiele}@tik.ee.ethz.ch

Abstract. Sensor network MAC protocols are typically configured for an intended deployment scenario once and for all at compile time. This approach, however, leads to suboptimal performance if the network conditions deviate from the expectations. We present ZeroCal, a distributed algorithm that allows nodes to dynamically adapt to variations in traffic volume. Using ZeroCal, each node autonomously configures its MAC protocol at runtime, thereby trying to reduce the maximum energy consumption among all nodes. While the algorithm is readily usable for any asynchronous low-power listening or low-power probing protocol, we validate and demonstrate the effectiveness of ZeroCal on X-MAC. Extensive testbed experiments and simulations indicate that ZeroCal quickly adapts to traffic variations. We further show that ZeroCal extends network lifetime by 50% compared to an optimal configuration with identical and static MAC parameters at all nodes.

1 Introduction

The medium access control (MAC) protocol is the core component of the sensor network protocol stack. It is responsible for turning the radio device on and off at regular intervals. This duty-cycling functionality entails a fundamental trade-off between energy consumption and bandwidth: While the radio should be asleep as much as possible to save energy, sufficient bandwidth must be provided to achieve a target delivery rate. Numerous MAC protocols have been devised [5], but up to date it is unclear how to configure these protocols.

One possible approach is to define a network-wide trade-off at compile time; that is, the parameters of the MAC protocol are the same for all nodes and remain unchanged once the sensor network has been deployed. We call this an *identical MAC configuration*. Finding such a configuration is a nontrivial task, as it requires detailed knowledge about the protocol and the network conditions. Moreover, an identical configuration does not account for different traffic volumes at different nodes, *e.g.*, increased routing traffic towards the sink. To illustrate this consider Figure 1(b), showing the average power consumption of four different MAC configurations with identical parameters for the same topology and data rate. While the energy consumption varies considerably for different configurations, we also see that nodes closer to the sink consume more energy as they carry a higher load. With an identical MAC configuration the sink's one-hop neighbors will potentially run out of energy long before the other nodes.



(a) Topology (partial) (b) Identical parameter configurations (c) ZeroCal

Fig. 1. The MAC configuration with identical parameters results either in a very high or imbalanced power consumption. ZeroCal instead adapts the sleep interval T_s to minimize the maximum, which results in a well-balanced average power consumption. Results from testbed experiments with 21 nodes running X-MAC (see Section 5).

These energy bottlenecks greatly reduce the *network lifetime*, which is defined as the time until the first node runs out of energy. Furthermore, sensor networks operate in very dynamic environments. Channel characteristics change due to varying environmental conditions, traffic volume increases or decreases with the frequency of change in the sensed phenomena, and nodes are potentially added or removed. These dynamics are likely to render an identical MAC configuration inefficient over time, suggesting that nodes should continuously adapt their settings to maintain satisfactory performance.

We present ZeroCal, a Zero Configuration algorithm. ZeroCal is a distributed algorithm that automatically configures the MAC protocol at runtime according to the current traffic volume and network conditions (*e.g.*, packet loss, interference, and network topology). Using ZeroCal, each node decides autonomously on its own parameter setting, thereby trying to increase the network lifetime (*i.e.*, reduce the maximum energy consumption among all nodes). The algorithm is based on the observation that a node's MAC configuration does not only influence its own energy consumption but also the one of its children. For example, a node can decide to spend more energy for receiving messages (by waking up more often) in exchange for a reduced transmission energy at its child nodes.

ZeroCal tries to extend network lifetime. Hence, it looks at the maximum energy consumption of a node and its children: The MAC parameters are set such that this maximum is minimized. This results in a well-balanced energy consumption across the network, as shown in Figure 1(c). ZeroCal's configuration process repeats regularly and adapts to changes in traffic volume, independent of whether the change is due to a higher packet loss along an incoming link, an increased sending rate, or a change in the routing topology.

We make the following contributions: *i)* We present ZeroCal, a distributed algorithm that automatically configures the MAC protocol at runtime. Assuming many-to-one data traffic, ZeroCal is readily usable for any asynchronous low-power listening or low-power probing protocol. *ii)* We validate ZeroCal on a testbed and in simulation, using X-MAC [2] as a case study. *iii)* We demonstrate that ZeroCal quickly adapts to traffic variations and extends network lifetime by 50% compared to an optimal, identical MAC configuration.

The remainder of this paper is organized as follows: Section 2 provides more background on sensor network MAC protocols and reviews related work on MAC protocol adaptation. Section 3 presents the design and underlying models of ZeroCal, which we validate in simulation in Section 4 and demonstrate on a testbed in Section 5. We conclude the paper in Section 6.

2 Background and Related Work

Low-power operation and energy management are of utmost importance. By putting the radio of a sensor node into sleep mode, the energy consumption can be reduced by orders of magnitude from mA to µA. Since the radio's wake-up time is rather short (<2 ms), it is possible to wake up the radio for only a very short time to exchange messages. This duty cycling is the primary task of the MAC protocol. A multitude of duty-cycling MAC protocols for sensor networks exist [5]. In the following, we briefly discuss two of the most prominent classes: *i)* low-power listening (LPL) and *ii)* low-power probing (LPP). Both Contiki and TinyOS feature default MAC protocols based on these two approaches. TDMA-based and slotted protocols are described elsewhere (*e.g.*, in [5]), and are outside the scope of this paper.

Using LPL nodes sleep most of the time and wake up regularly to quickly poll the channel. If a node detects a carrier, it keeps its radio on to receive a message; otherwise, it goes back to sleep. As nodes wake up asynchronously, the sender must transmit a preamble for a period slightly exceeding the sleep interval so that the receiver can detect the carrier. In X-MAC [2] the preamble is a sequence of short advertisement packets that contain the address of the receiver, as shown in Figure 2. After sending an advertisement, the sender listens for an acknowledgment from the receiver. If the sender hears an acknowledgment, it sends the data packet. LPP-based protocols (*e.g.*, RI-MAC [10]) take the inverse approach. Instead of polling the channel, nodes send an announcement when they are awake and subsequently listen for a data packet. The sender must wait for such an announcement from the intended receiver before it can send the data packet. LPP occupies the channel less than LPL because no long preambles are transmitted. The active period, however, is longer with LPP as it must accommodate an announcement and an incoming message.

LPL and LPP have one parameter in common: the sleep interval T_s. This is the time the radio is put to sleep between two active periods. The sleep interval greatly influences the average power consumption of a node, as illustrated in Figure 1(b). A short sleep interval reduces the energy consumption for sending messages but results in an increased energy consumption for polling the channel. Moreover, the sleep interval determines the available bandwidth. In fact, there exists an optimal sleep interval for a given traffic volume that minimizes energy consumption and provides sufficient bandwidth.

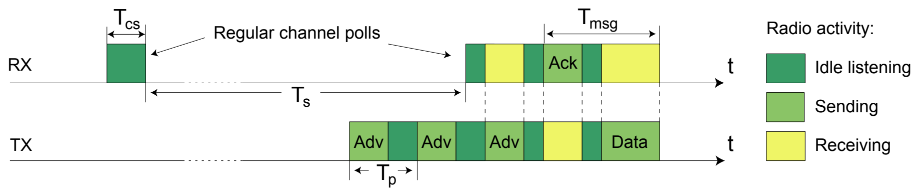


Fig. 2. Basic concept of LPL using the example of X-MAC. The receiver is sleeping most of the time, waking up every sleep interval T_s to poll the channel for T_{cs}. The sender transmits consecutive advertisement packets, waiting after each of them for an acknowledgment from the receiver that allows it to send the data packet.

Polastre *et al.* [9] discuss the benefit of adapting B-MAC based on varying network conditions. They derive an analytical model of node lifetime and argue that other services could use this model to recompute check interval and preamble length. ZeroCal provides such a service and uses an energy model to optimize and adapt the MAC configuration as traffic volume changes.

Buettner *et al.* [2] adapt the sleep interval of X-MAC for a single sender-receiver pair. Using the estimated probability of receiving a packet, the receiver chooses its sleep interval such that the sum of transmit and receive energy is minimized. In contrast, ZeroCal works on any mesh or tree topology and respects the influence on the sender when choosing a new sleep interval at the receiver. More importantly, ZeroCal ensures that the sender's preamble is long enough so that it can be detected by the receiver. This reliability issue is inherent if nodes adapt their sleep interval autonomously but is not addressed in [2].

Jurdak *et al.* [4] propose a cross-layer framework for network-wide energy optimization and load balancing through greedy local decisions. We show that it is indeed beneficial not to optimize energy in a greedy fashion, but to consider both parent and children in the optimization. Merlin *et al.* [7] present a control-theoretic approach to adapt the duty cycle, which is however only suited for single-hop topologies.

3 ZeroCal

Asynchronous low-power listening and low-power probing MAC protocols feature a trade-off between bandwidth and energy consumption for a specific node and all nodes communicating with it. In data collection, one can either have the parent node spending a lot of energy for idle listening or the child nodes for sending messages. ZeroCal extends network lifetime by minimizing the maximum energy consumption of each parent-children pair in the network. To this end, ZeroCal optimizes the sleep interval of a parent at runtime based on its traffic volume. Since every parent is also a child (except for the sink), this adaptive approach results in a well-balanced energy consumption across the whole network.

ZeroCal adapts the MAC configuration as illustrated in Figure 3. The optimization uses an energy model (see Section 3.3) that is based on records collected

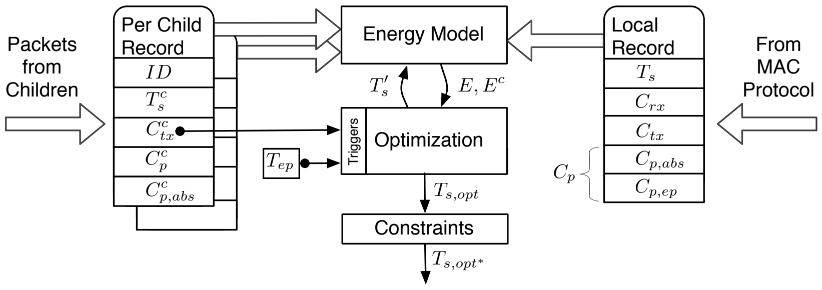


Fig. 3. ZeroCal architecture. ZeroCal keeps records of local and child MAC parameters. Periodically, an optimization process is triggered. Using an energy model, a new optimal sleep interval $T_{s,opt*}$ is computed respecting bandwidth and protocol constraints.

from the local MAC and the child nodes (see Section 3.2). The optimization explores the effect of a new sleep interval T_s' on the maximum energy consumption of the node-children pair and returns an optimal sleep interval $T_{s,opt}$. Furthermore, the MAC protocol itself imposes certain constraints on its parameters, which require to adapt $T_{s,opt}$ to $T_{s,opt*}$.

ZeroCal performs the optimization over time windows, called *epochs*. The optimization is triggered when either the transmission count of a child node C_{tx}^c exceeds the threshold C_{eval} or the *epoch time* T_{ep} exceeds the maximum duration of an epoch $T_{ep,max}$. The traffic-dependent trigger is needed to react to a (sudden) increase in traffic volume, whereas the timed trigger ensures that ZeroCal periodically updates the MAC configuration when only few or no messages arrive, *e.g.*, at leaf nodes.

3.1 Parameter Optimization

We now describe how ZeroCal uses the energy estimation to determine an optimal MAC configuration at runtime.

In general, a node can save energy by increasing the length of its sleep interval T_s. Indeed, it could sleep as long as possible to maximize its own lifetime. Such an approach is however very selfish, since it increases the energy consumption for sending messages at child nodes. This raises the question whether it is more important to save energy for oneself or at the child nodes. Since the goal is to prolong the lifetime of both the parent and its children, ZeroCal chooses the sleep interval $T_{s,opt}$ at the parent such that the maximum of the parent's energy consumption E and of its children E^c is minimized:

$$T_{s,opt} = \underset{T_s' \in [T_{s,min}, T_{s,max}]}{\operatorname{argmin}} \max \left[E, \max_{\forall \text{ children } c} (E^c)\right]. \quad (1)$$

Additionally, ZeroCal has to ensure that the parent is able to detect the preamble of its children. Therefore, the parent's sleep interval $T_{s,opt*}$ must be chosen to be shorter or equal to the longest sleep interval of its child nodes:

$$T_{s,opt*} \leq T_s^c, \quad \forall \text{ children } c. \quad (2)$$

Moreover, the sleep interval also limits the available bandwidth at a node. Therefore, it might be necessary to further reduce the sleep interval $T_{s,opt*}$ to increase the bandwidth. For instance, by requiring

$$T_{s,opt*} \leq 1/(C_{tx} + C_{rx})/n \,, \tag{3}$$

we ensure that at most in every n-th sleep interval a packet is being sent or received.

3.2 Collecting MAC Statistics

To compute up-to-date energy estimates during the optimization task, each node keeps a record of counters and sleep intervals of itself and all its children, as shown in Figure 3. Locally the following information is available: The number of sent C_{tx} and received messages C_{rx}, the current sleep interval T_s, and the epoch time T_{ep}. The number of transmitted preamble packets C_p is determined by the absolute preamble count $C_{p,abs}$ in reference to the one at the beginning of the epoch $C_{p,ep}$ via $C_p = C_{p,abs} - C_{p,ep}$.

From the child node the number of sent messages C^c_{tx} is readily available as it corresponds to the number of messages received from the child. Here, we neglect lost data packets along incoming links. We further simplify and set $C^c_{rx} = C^c_{tx}$, approximating that child nodes are pure forwarders and do not generate messages themselves. The sleep interval T^c_s and the absolute preamble count $C^c_{p,abs}$ are piggybacked on data packets. Since we are interested in the number of preamble packets sent during the current epoch, we additionally maintain the preamble count C^c_p that is updated at every packet reception: Assuming a node received the k-th packet in an epoch (*i.e.*, the received absolute value is $C^c_{p,abs}(k)$ and the previous one locally stored is $C^c_{p,abs}(k-1)$), we can determine the difference $\Delta C^c_p = C^c_{p,abs}(k) - C^c_{p,abs}(k-1)$ and subsequently update the preamble count $C^c_p(k) = C^c_p(k-1) + \Delta C^c_p$. After running the optimization, a node resets the epoch time and all counters, except for its local preamble count $C_{p,abs}$ which is stored in $C_{p,ep}$. The overhead for collecting estimation information is low. In our current implementation, a parent reserves 9 bytes per child, and 3 additional bytes are piggybacked on each data packet.

3.3 Energy Model

ZeroCal needs to estimate a node's energy consumption given its MAC configuration. In this work, we use a refined estimation method based on our previous work [6]. We only consider the energy consumption of the radio device, which is a reasonable assumption since the radio is usually the main consumer of energy in the system.

The radio is active while performing regular channel polls T_{cp}, transmitting messages T_{tx}, and receiving messages T_{rx}. We opt to neglect the effect of interference, assuming that the impact on the overall energy budget is minimal compared to T_{cp}, T_{tx}, and T_{rx}. Given these times and the corresponding average power consumptions, we can estimate a node's energy consumption using

$$E = T_{tx} \cdot P_{tx} + T_{rx} \cdot P_{rx} + T_{cp} \cdot P_{cp} \,. \tag{4}$$

We note that P_{tx}, P_{rx}, and P_{cp} are not the power levels of the radio device in transmit, receive, and idle mode. Instead, they correspond to the average power consumption of the logical states of the MAC protocol: sending, receiving, and channel polling. For instance, while sending a message (preamble stream, acknowledgment reception, plus data packet transmission), the radio switches several times between transmit and receive mode. The average power levels depend on hardware platform and protocol implementation, and are measured offline.

We estimate the residence times in the different protocol states using the sleep interval T_s, the number of received C_{rx} and sent messages C_{tx}, and the number of transmitted preamble packets C_p:

$$\begin{aligned} T_{tx} &= C_p \cdot T_p + C_{tx} \cdot T_{msg} \,, \\ T_{rx} &= C_{rx} \cdot T_{msg} \,, \\ T_{cp} &= (T_{ep} - T_{tx} - T_{rx}) \cdot T_{cs}/(T_s + T_{cs}) \,, \end{aligned} \tag{5}$$

where T_p, T_{cs}, and T_{msg} are constants specific to the radio device and the MAC protocol as illustrated in Figure 2. The time spent for sending messages depends on the number of transmitted preamble packets and the number of sent data packets. Sending a data packet takes T_{msg}, which includes the actual data packet, its acknowledgment, and the radio switching times. For every received message, the radio is active for T_{msg}. If there is no traffic, nodes wake up regularly every T_s and go back to sleep after T_{cs}. The length of T_{cs} also includes the time for switching the radio between sleep and receive mode.

The energy estimation according to Equation 5 depends on the current sleep interval T_s; choosing a new sleep interval T_s' affects the times spent in the different logical states of the MAC protocol at both the parent and its child nodes. Assuming a linear relation between the number of preamble packets and the parent's sleep interval, ZeroCal estimates the new times as follows:

– At the parent, the times for receiving T_{rx} and transmitting T_{tx} do not change, whereas the new channel-polling time T_{cp} is given by

$$T_{cp} = (T_{ep} - T_{tx} - T_{rx}) \cdot T_{cs}/(T_s' + T_{cs}) \,. \tag{6}$$

– At the child nodes, only the time for transmitting messages T_{tx}^c is affected:

$$T_{tx}^c = C_p^c \cdot T_p^c \cdot T_s'/T_s + C_{tx}^c \cdot T_{msg} \,. \tag{7}$$

To validate the accuracy of our modeling, we run simulations in Castalia [8] on a binary tree topology (see Section 4 for details) and compare the estimated energy with the energy measured by the simulator. We observe that for both low and high data rates the energy consumption is well estimated across all simulated

MAC configurations: The maximum estimation error ranges between 1.8% (low data rate) and 4.1% (high data rate).

Our energy estimation model is general enough to accommodate other MAC protocols as well. For example, to adapt the model to a LPP-based protocol (*e.g.*, RI-MAC [10]), we have to replace the preamble counter C_p in Equation 5 with a counter that keeps track of the number of time intervals (with length T_p) a node waits for an announcement from the receiver.

3.4 System Integration

As for the routing, we assume many-to-one data traffic that flows toward a common sink node. On each intermediate node, messages are forwarded to a parent that is closer to the sink with respect to some routing metric (*e.g.*, hop count or ETX [3]). Our approach works on any mesh and tree topology. Furthermore, we do not require a dedicated interface between the MAC and routing layers. However, we assume that unicast transmissions take place only between a child and its parent. ZeroCal optimizes for the current amount of traffic, which typically differs across the network. Hence, ZeroCal's operation is independent of whether traffic originates from local data generation or packet forwarding.

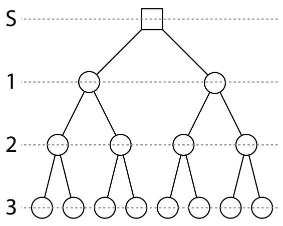
In sensor networks, the sink node typically has unlimited power supply from a base station, which is responsible for additional tasks, such as providing a back channel via GPRS to a central monitoring system [1]. This setup allows for a so-called *always listening sink* that runs a 100% duty cycle. ZeroCal adapts to this scenario: The always listening sink reduces the average transmission time of its one-hop neighbors to a minimum, and the repeated execution of the algorithm propagates this benefit down to the leaf nodes, eventually reducing the energy consumption of all nodes.

4 Simulation

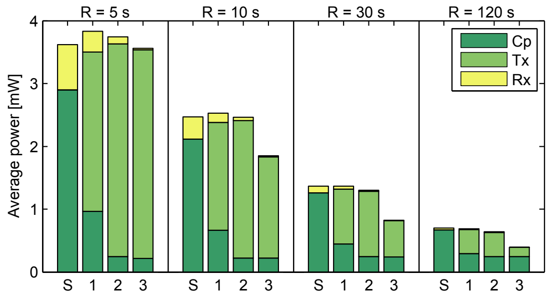
We implement ZeroCal and X-MAC in Castalia [8], a state-of-the-art sensor network simulator. Castalia features a detailed model of the radio device that also accounts for the transition times between the different operational modes and their individual power consumptions. Furthermore, Castalia provides a realistic model of the wireless channel with random packet loss and interference.

For illustration purposes, we use a perfect binary tree of depth 3 in our simulations, as shown in Figure 4(a). In the binary tree, we have one sink with two children (level 1), four grandchildren (level 2), and eight leaf nodes (level 3). Child nodes interfere with each other, and a grandparent is a hidden terminal for a child. We use Castalia's CC2420 radio model and set the packet reception rate of all links to 90% to see whether ZeroCal can cope with packet loss.

Every node but the sink samples and generates data packets with an inter-packet interval R, ranging from 5 s (high data rate) to 120 s (low data rate). To study the long-term behavior of ZeroCal, we run simulations that correspond to



(a) Simulation topology



(b) Average power consumption with ZeroCal

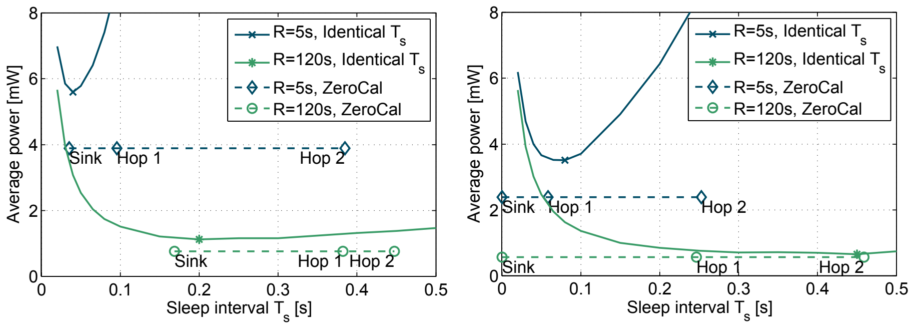
Fig. 4. Simulation with binary tree topology: ZeroCal shows a well balanced average power consumption within the network. For every hop, the node with the highest average power consumption is depicted.

one day in real time. We use the following inputs for the parameter optimization: $T_{s,\min} = 20\,\text{ms}$, $T_{s,\max} = 500\,\text{ms}$, $T_{ep,max} = 500\,\text{s}$, $C_{eval} = 50$ packets, and $n = 3$ sleep intervals. Unless otherwise stated, the sink is duty cycled.

4.1 Adaptive Behavior

We first look at the adaptive behavior of ZeroCal for different traffic volumes. Figure 4(b) plots for each level in the binary tree (*i.e.*, hop-distance from the sink) the maximum energy consumption. We see that ZeroCal achieves a well-balanced energy consumption across all levels; the remaining differences are due to the upper bound on the sleep interval ($T_{s,max} = 500\,\text{ms}$). For instance, at a sampling interval of $R = 5\,\text{s}$ the energy consumptions differ only by 7.5% *among all nodes*. This demonstrates ZeroCal's capability to evenly distribute the workload across the entire network.

In fact, ZeroCal gradually aligns the energy consumptions by distributing the *type* of workload. To see this, we take a closer look at the composition of energy consumption at different levels in Figure 4(b): The portion of transmit (Tx) energy increases with hop-distance, whereas the portions of receive (Rx) and channel-polling (Cp) energy decrease. For example, the nodes on level 3 spend relatively more transmit energy than the nodes on level 1, even though they send seven times fewer messages. The rationale behind this behavior is justified by the following reasoning. The sink does not send any messages and can therefore invest more energy into frequent wake-ups and receiving ($T_s^S = 35\,\text{ms}$). This in turn reduces the transmission energy for the nodes on level 1, which choose a shorter sleep interval ($T_s^1 = 96\,\text{ms}$) than the nodes on level 2 ($T_s^2 = 385\,\text{ms}$). Finally, the nodes on level 3 select the longest possible sleep interval ($T_s^3 = 500\,\text{ms}$). ZeroCal's periodic parameter optimization results in a step-by-step adaptation toward this optimal point of operation, as changes in the MAC configuration on one level eventually propagate to all other levels in the tree.



(a) Duty-cycled sink (b) Always listening sink

Fig. 5. Average power consumption for the network-wide, static (solid lines) and ZeroCal's adaptive (dashed lines) sleep interval. ZeroCal outperforms all static configurations by choosing different sleep intervals at different hop-distances.

4.2 Static versus Adaptive Configuration

We now want to quantify the benefit of ZeroCal. What do we gain from the adaptation as compared to an identical MAC configuration?

With an identical configuration, we face the problem of finding the optimal MAC parameters in advance, which depend on application and deployment characteristics, such as sampling interval, routing topology, and interference. The best we can do is to carry out preliminary experiments on the deployment site to make an educated guess on these ever-changing variables.

Figure 5(a) highlights the importance of choosing the right configuration. The solid lines show the average power consumption as a function of the sleep interval for both a high ($R = 5\,\text{s}$) and a low data rate ($R = 120\,\text{s}$), again on the perfect binary tree. The optimal sleep intervals for an identical configuration correspond to the minima indicated by markers. We see that the curve for the high data rate is very steep; that is, our system would be very inefficient for high traffic even if we are only slightly off the optimal sleep interval. This dependency is more moderate for low traffic. Additionally, we have to consider that the sleep interval limits the available bandwidth. For instance, if we choose a sleep interval longer than 100 ms for the high data rate, packets are potentially lost as nodes are overloaded by incoming traffic. Overall, finding a suitable identical configuration is critical and hard to achieve.

ZeroCal outperforms all identical MAC configurations in our simulations. This can also be seen in Figure 5, where we show ZeroCal's maximum energy consumption with a dashed line. Furthermore we indicate the average sleep intervals chosen by ZeroCal at each hop-distance. For a duty-cycled sink in Figure 5(a), the adaptive sleep interval reduces the maximum energy consumption by 30.6% ($R = 5\,\text{s}$) and 32.7% ($R = 120\,\text{s}$) compared with the corresponding optimal, identical configuration. If we have an always listening sink, we see in Figure 5(b) that ZeroCal adapts well to this situation: the energy savings are 32.0% ($R = 5\,\text{s}$)

and 13.3% ($R = 120\,\mathrm{s}$) in comparison to the static configuration. We note again that ZeroCal distributes the workload evenly, avoiding energy hot spots that would otherwise limit the network lifetime.

Looking at the sleep intervals chosen by ZeroCal, we observe that they are shorter for an always listening sink than for a duty-cycled sink (*i.e.*, nodes save energy while polling the channel more frequently). This might surprise at first, but is explained by the fact that the always listening sink allows the sink's one-hop neighbors to save a lot of transmit energy. ZeroCal's periodic adaptation propagates these energy savings eventually to the nodes farther away, which can then reduce their sleep interval to wake up more frequently.

4.3 Large and Irregular Topologies

We also perform simulations on large and irregular topologies to see whether ZeroCal scales. ZeroCal automatically optimizes for the subtree with the highest data load: the corresponding nodes show a well-balanced energy consumption and hence the network lifetime is maximized. The nodes in subtrees with less traffic also balance their energy consumption, yet on a lower energy level.

For instance, with 100 nodes on an area of 160 by 160 meters and up to 8 hops to the sink, we observe the same trend as with the binary tree topology. For high, medium, and low data rates, ZeroCal reduces the maximum energy consumption by 28.7%, 33.7%, and 33.5% compared to an optimal, identical MAC configuration. The energy consumption averaged over all nodes is reduced by 61.7%, 55.3%, and 51.2%, which indicates that ZeroCal optimizes the energy consumption for nodes with little traffic even further.

5 Testbed Experiments

To demonstrate the effectiveness of ZeroCal on real sensor nodes, we implement ZeroCal in Contiki on top of X-MAC. We run experiments on a testbed of 21 Tmote Sky nodes deployed over several offices. We use an irregular tree topology with a maximum hop distance to the sink of 5, as shown in Figure 1(a). We fix the topology to be able to compare the results of different runs. However, there are two types of network dynamics that affect ZeroCal's operation: *i)* varying packet loss due to various sources of interference (*e.g.*, WLAN and moving people), and *ii)* varying traffic volume as nodes change their sampling rate at runtime.

ZeroCal performs an exhaustive parameter search on the sleep interval during the parameter optimization, using the same settings as in our simulations. To speed up the process at nodes with high incoming degree, ZeroCal considers only two of their children during the search: the one with the lowest sleep interval and the one with the highest preamble count. These two nodes are likely to carry the highest energy load among all children and thus matter most when minimizing the maximum energy consumption. Furthermore, ZeroCal uses an adaptive granularity during the search. For a sleep interval close to the minimum $T_{s,min} = 20\,\mathrm{ms}$ ZeroCal makes steps of 4 ms, whereas for a sleep interval close to the maximum $T_{s,max} = 500\,\text{ms}$ it evaluates in steps of 30 ms. We find that the computational speed-up outweighs the slight loss in accuracy.

5.1 Static versus Adaptive Configuration

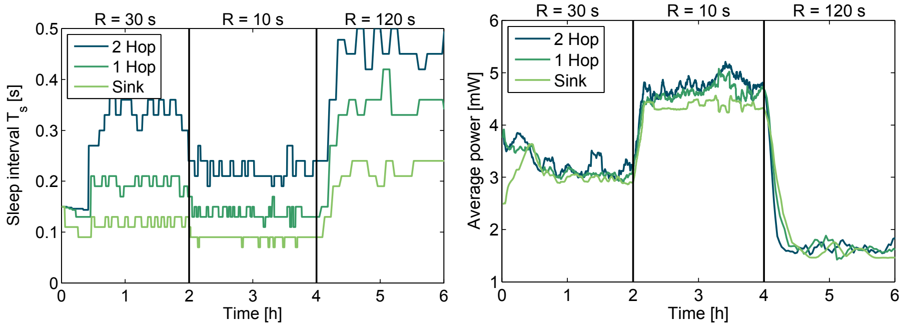
In a first series of experiments, we compare ZeroCal with an identical X-MAC configuration. All nodes generate data packets with a constant inter-packet interval of $R = 30\,\text{s}$. As shown in Figure 1, ZeroCal outperforms the optimal, identical setting ($T_s = 200\,\text{ms}$) by 32.6%, which translates in an increase of 48.4% in network lifetime. We see that ZeroCal achieves a well-balanced energy consumption across all hop-distances. This indicates that ZeroCal configures X-MAC nearly optimal. We also see in in Figure 1(b) that Contiki's default X-MAC configuration ($T_s = 500\,\text{ms}$) can lead to poor performance. If an inexperienced user simply uses the default settings, radio communication requires 60.8% more energy in comparison to ZeroCal's adaptive configuration.

5.2 Adaptation to Network Dynamics

Finally, we analyze how ZeroCal adapts to varying network conditions, which is impossible for an identical MAC configuration. To this end, we let nodes change their sampling interval dynamically every two hours. Starting with a medium rate ($R = 30\,\text{s}$), nodes switch after two hours to a high rate ($R = 10\,\text{s}$), followed by another change to a low rate ($R = 120\,\text{s}$) after four hours. At the same time, sporadic packet losses occur.

Figure 6(a) shows how ZeroCal adapts X-MAC's sleep interval at different hop-distances over time, and Figure 6(b) shows the corresponding trend in the average power consumption. At the beginning, all nodes have the same sleep interval ($T_s = 150\,\text{ms}$). While the sink quickly adapts to a stable T_s, it takes about 25 minutes until the two-hop neighbors adapt their T_s. This is because a node is not allowed to have a longer T_s than its children, as enforced by Equation 2. Hence, the adaptation from the initial (far from optimal) T_s toward a longer T_s starts at the leaf nodes and then propagates step-by-step upward in our tree topology of depth 5 (see Figure 1(a)). Conversely, adapting to the high traffic volume after two hours is very fast. First, the optimization task is performed more often at high data rates and, second, there are no constraints when reducing T_s. Note that the adaption latency mainly depends on the duration of an epoch, which we set conservatively ($T_{ep,max} = 500\,\text{s}$) to avoid system instabilities. The variations in T_s after adapting to a new data rate are mainly inflicted by external effects, such as packet loss and interference.

Looking at the evolution of power consumption in Figure 6(b), we note that ZeroCal distributes the load evenly as network conditions change. Only at $R = 10\,\text{s}$, we note a 7% difference in energy consumption between the sink and the other nodes. This is because our energy estimation model ignores effects like overhearing and collisions, which happen more frequently at high data rates. However, as sensor networks typically sample at low data rates, we prefer a simpler model inducing less overhead. When the traffic volume drops, there is a slight adaptation latency. More importantly, however, ZeroCal quickly adapts to a (sudden)



(a) Sleep interval (b) Average power over last 10 minutes

Fig. 6. Testbed experiments with changing data rates. ZeroCal adapts especially quickly if the data rate increases and bandwidth is required.

increase in traffic volume when additional bandwidth is needed while keeping the energy consumption as low as possible.

6 Conclusions

There has been considerable research on sensor network MAC protocols. Until now, their proper configuration has been mostly neglected. While network dynamics make it hard to define an appropriate configuration at deployment time, we showed that the MAC configuration has indeed a wide impact on energy consumption and available bandwidth. Particularly if expert knowledge is missing, the MAC protocol is often used with its default parameters, which can result in very poor energy efficiency. In the worst case, application-specific bandwidth and lifetime requirements are not satisfied.

To tackle these challenges, we proposed ZeroCal, a distributed algorithm that automatically configures the MAC parameters at runtime in order to increase the network lifetime. The configuration is repeated regularly and ensures that the nodes adapt to variations in traffic volume.

We validated ZeroCal on X-MAC in simulation and testbed experiments. ZeroCal quickly adapts toward a stable operating point, in particular when additional bandwidth is needed. Moreover, ZeroCal outperforms an optimal, identical MAC configuration, extending network lifetime by about 50%.

Acknowledgements

The work presented in this paper was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. We thank Koen Langendoen and Thiemo Voigt for their feedback on draft versions of this paper.

References

1. Beutel, J., Gruber, S., Hasler, A., Lim, R., Meier, A., Plessl, C., Talzi, I., Thiele, L., Tschudin, C., Woehrle, M., Yuecel, M.: PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes. In: Proc. 8th Int'l Conf. Information Processing Sensor Networks (IPSN 2009), San Francisco, CA, USA, April 2009, pp. 265–276. ACM/IEEE (2009)
2. Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: Proc. 4th ACM Conf. Embedded Networked Sensor Systems (SenSys 2006), pp. 307–320. ACM Press, New York (2006)
3. Couto, D.S.J.D., Aguayo, D., Bicket, J., Morris, R.: A high-throughput path metric for multi-hop wireless routing. Wireless Networks 11(4), 419–434 (2005)
4. Jurdak, R., Baldi, P., Lopes, C.V.: Adaptive low power listening for wireless sensor networks. IEEE Transactions on Mobile Computing 6, 988–1004 (2007)
5. Langendoen, K.: Medium access control in wireless sensor networks. In: Wu, H., Pan, Y. (eds.) Medium Access Control in Wireless Networks, May 2008, pp. 535–560. Nova Science Publishers, Inc., Bombay (2008)
6. Langendoen, K., Meier, A.: Analyzing MAC protocols for low data-rate applications. ACM Transactions on Sensor Networks (2010) (accepted for publication)
7. Merlin, C.J., Heinzelman, W.B.: Duty cycle control for low-power-listening MAC protocols. In: 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2008, September/October 2008, pp. 497–502 (2008)
8. Pham, H.N., Pediaditakis, D., Boulis, A.: From simulation to real deployments in WSN and back. In: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2007, June 2007, pp. 1–6 (2007)
9. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), pp. 95–107. ACM Press, New York (2004)
10. Sun, Y., Gurewitz, O., Johnson, D.: RI-MAC: A receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In: Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003), Raleigh, NC, November 2008, pp. 1–14 (2008)

Programming Sensor Networks Using REMORA Component Model

Amirhosein Taherkordi[1], Frédéric Loiret[2], Azadeh Abdolrazaghi[1], Romain Rouvoy[1,2], Quan Le-Trung[1], and Frank Eliassen[1]

[1] University of Oslo, Department of Informatics
P.O. Box 1080 Blindern, N-0314 Oslo
{amirhost,azadeha,rouvoy,quanle,frank}@ifi.uio.no
[2] INRIA Lille – Nord Europe, ADAM Project-team,
University of Lille 1, LIFL CNRS UMR 8022,
F-59650 Villeneuve d'Ascq
{frederic.loiret,romain.rouvoy}@inria.fr

Abstract. The success of high-level programming models in *Wireless Sensor Networks* (WSNs) is heavily dependent on factors such as ease of programming, code well-structuring, degree of code reusability, and required software development effort. Component-based programming has been recognized as an effective approach to meet such requirements. Most of componentization efforts in WSNs were ineffective due to various reasons, such as high resource demand or limited scope of use. In this paper, we present REMORA, a new approach to practical and efficient component-based programming in WSNs. REMORA offers a well-structured programming paradigm that fits very well with resource limitations of embedded systems, including WSNs. Furthermore, the special attention to event handling in REMORA makes our proposal more practical for WSN applications, which are inherently event-driven. More importantly, the mutualism between REMORA and underlying system software promises a new direction towards separation of concerns in WSNs. Our evaluation results show that a well-configured REMORA application has an acceptable memory overhead and a negligible CPU cost.

Keywords: Wireless sensor networks, component model, event-driven.

1 Introduction

The recent increase in the number and size of WSN applications makes *high-level programming* an essential need to the development of WSN platforms. However, this concept is still immature in the context of WSNs for various reasons. Firstly, the existing diversities in WSN hardware and software platforms have brought the same order of diversity to programming models for such platforms [1]. Moreover, developers' expertise in state-of-the-art programming models become useless in WSN programming as the well-established discipline of program specification is largely missing in this area. Secondly, the structure of programming models for WSNs are usually sacrificed for resource usage efficiency, thereby, the outcome of such models is usually a piece of tangled code maintainable only by its owner. Finally, application programming in WSNs typically requires learning low-level system programming languages, which imposes a significant burden on the programmer.

Software *componentization* has been recognized as a well-structured programming model able to tackle the above concerns. Separation of concerns, module reusability, controlling cohesion and coupling, and provision of standard API are some of the main features of *component-based software engineering* [2,3]. Although using this paradigm in earlier embedded systems was relatively successful [4,5,6,7], most of the efforts in the context of WSNs remain inefficient or limited in the scope of use. TINYOS programming model, NESC [8], is perhaps the most popular component model for WSNs. Whereas NESC eases WSN programming, this component model is tightly bound to the TINYOS platform. Other proposals, such as OPENCOM [14] and THINK [20], are either too heavyweight for WSNs, or not able to support event-driven programming, which is of high importance in WSNs.

In this paper, we present REMORA, a lightweight component model designed for resource-constraint embedded systems, including WSNs. The strong abstraction promoted by this model allows a wide range of embedded systems to exploit it at different software levels from *Operating System* (OS) to application. To achieve this goal, REMORA provides a very efficient mechanism for event management, as embedded applications are inherently event-driven. REMORA components are described in XML as an extension of the *Service Component Architecture* (SCA) model [10] in order to make WSN applications compliant with the state-of-the-art componentization standards. Additionally, the C-like language for component implementation in REMORA attracts both embedded system programmers and PC-based developers to programming for WSNs. Finally, REMORA features a coherent mechanism for component *instantiation* and *property-based component configuration* in order to facilitate lightweight event-driven programming in WSNs.

We demonstrate the promising result of deploying REMORA components on Contiki—a leading operating system for WSNs [11]. The efficient use of Contiki features, such as process management and event distribution [12], on the one hand, and the abstraction layer linking REMORA to Contiki, on the other hand, promise a very effective and generic approach towards practical high-level programming in WSNs.

The rest of the paper is organized as follows. In Section 2, the specification of the REMORA component model is presented. Section 3 describes how REMORA is implemented, while the evaluation results are reported in Section 4 including the assessment of a real REMORA-based deployment. A survey of existing approaches and a discussion on REMORA future work are presented in Section 5 and Section 6, respectively.

2 REMORA Component Model

In this section, we first discuss the primary design concepts in REMORA and then we explain the specifications of this component model. The design principles of REMORA include:

XML-based Component Description. To achieve simplicity and generality, we adopt XML to describe components. The XML schema in REMORA conforms to the *Service Component Architecture* (SCA) notations in order to accelerate standardization of component-based programming in WSNs. As SCA is originally designed for large-scale systems-of-systems, REMORA extends SCA with its own architectural concerns to achieve realistic component-based programming in WSNs.

C-like Language for Component Implementation. REMORA components are written in a C-like language enhancing the C language with features to support component-based and structured programming. This enhancement also attracts both embedded systems programmers and PC-based developers towards high-level programming in WSNs.

OS Abstraction Layer. The REMORA component framework is integrated with underlying operating system through a well-defined OS-abstraction layer. This thin layer can easily be developed for all WSN operating systems supporting the C language like Contiki. This feature ensures portability of REMORA components towards different OSs. The abstraction of REMORA becomes more valuable when the component framework is easily configured to reuse OS-provided features, such as event processing and task scheduling.

Event Handling. Besides the support of events at operating system level in embedded systems, we also need to consider event handling at the application layer. REMORA proposes a high-level support of event generation and event handling. Indeed, the event-processing model of REMORA is one of its key features.

To describe our component model, we first define the basic terms used throughout this paper. Figure 1 illustrates the development process of REMORA-based applications. A REMORA application consists of a set of REMORA Components, containing descriptions and implementations of software modules. The REMORA *engine* processes the components and generates standard C code deployable within the REMORA *framework*. The framework is an OS-independent module supporting the specification of the REMORA component model. Finally, the REMORA application is deployed on the target sensor node through the REMORA *runtime*, which is an OS-abstraction layer integrating the application to the system software.

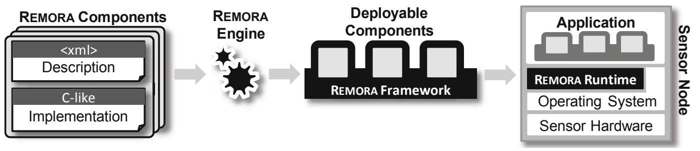


Fig. 1. Development process of REMORA-based applications

2.1 Component Specification

A REMORA component contains two main artifacts: component *description* and component *implementation*. The component description is an XML document describing the specifications of the component including *services*, *references*, *producedEvents*, *consumedEvents*, and *properties*. A service describes the operations provided by the component, while a reference indicates the operations required by the component. Likewise, a producedEvent identifies an event type generated by a component, whereas a consumedEvent specifies component's interest on receiving a particular event. The component implementation is a C-like program containing three types of operations: *i)* operations implementing the component's services, *ii)* operations processing events, and *iii)* component's private operations.

To overview the REMORA specification, we first present the REMORA-based implementation of the traditional *blink* application, then we discuss REMORA features in details. Figure 2 depicts the components involved in this application which are in charge of *blinking* a LED on sensor node every three seconds.

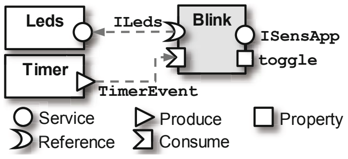


Fig. 2. A simple REMORA-based application

We here focus on the Blink component and describe it according to the REMORA component model. Figure 3 shows the XML description of this component. Blink provides an `ISensorApp` interface to start application execution and requires an `ILeds` interface to switch LEDs on and off, which is implemented by the Leds component. It also exposes a property to `toggle` a LED on the sensor node. As Blink produces no event, the `producer` tag is empty, while it is subscribed to receive TimerEvent and process it in the `timerExpired` function.

Figure 4 presents the excerpt of the Blink implementation. This C-like code implements the only function of the `ISensorApp` interface (`runApplication`) and handles TimerEvent within the `timerExpired` function. In the `runApplication` function, we specify that the TimerEvent generator (`aTimeEvent.producer`) is configured to generate periodically TimerEvent every three seconds. The last command in this function is used to notify the TimerEvent generator to start time measurement. When time is expired, Timer sets the attributes of aTimeEvent (*e.g.*, latency) and then the REMORA framework calls the `timerExpired` function.

Services and References. Components offer their function as *services* and may also depend on services provided by other components, so called *references*. A service consists of an *interface*, described in a separate XML with a name and the associated operations. Figure 5 presents the simplified `ILeds` interface used by the Blink component as a reference.

Component Properties. Properties are the editable parameters provided by each component, converting components from a dead unit of functionality to an active entity tractable during the application lifespan. In particular, this enhancement occurs in event producer components, where we need to retain the state of the event producer to generate accurate events, *e.g.*, the Timer component in the Blink application. Properties also enable components to become either *stateless* or *stateful*. A component is stateful if and only if it defines a property, *e.g.*, the Blink component is stateful, while Leds is a stateless component.

Component Implementation. REMORA components are implemented by using a dialect of C language with a set of new commands. This C-like language is mainly proposed to support the unique characteristics of REMORA, namely, component instantiation, event processing, and property manipulation. Therefore, for pure component-based programming without the above features, the programmer can almost rely on C features. We implicitly introduced a few of these commands within the Blink component implementation, while the complete description of commands is available in [22].

```
<componentType name="app.BlinkApp">
  <service name="iSensorApp">
    <interface.remora name="core.boot.api.ISensorApp"/>
  </service>
  <reference name="iLeds">
    <interface.remora name="core.peripheral.api.ILeds"/>
  </reference>
  <property name="toggle" type="xsd:short">0</property>
  <producer/>
  <consumer operation="timerExpired">
    <event.remora type="core.sys.TimerEvent" name="aTimeEvent"/>
  </consumer>
</componentType>
```

Fig. 3. XML description of Blink component

```
void runApplication(){
  aTimeEvent.producer.configure(3*CLOCK_SECOND, 1/*periodic*/);
  aTimeEvent.observation.start();
}
void timerExpired(){
  if (this.toggle == 0){
    iLeds.onLeds(LEDS_RED);
    this.toggle = 1;
  }else{
    iLeds.offLeds(LEDS_RED);
    this.toggle = 0;
  }
  printf("Time elapsed after interval: %d", aTimeEvent.latency);
}
```

Fig. 4. C-like implementation of Blink component

```
<interface.remora name="core.peripheral.api.ILeds">
  <operation name="getLeds" return="xsd:unsignedByte"/>
  <operation name="onLeds">
    <in name="leds" type="xsd:unsignedByte"/>
  </operation>
  <operation name="offLeds">
    <in name="leds" type="xsd:unsignedByte"/>
  </operation>
</interface.remora>
```

Fig. 5. A simplified description of `ILeds` interface

2.2 Component Instantiation

Component instantiation is essentially proposed to manage efficiently event producer components. The REMORA engine greatly benefits from component instantiation when linking one producer to several consumers. For example, in the Blink application, the producer (Timer) of TimerEvent should be instantiated per consumer component, while the UserButtonEvent generator is a single-instance component publishing an event to all subscribed components when the user button on a sensor node is pressed.

Component instantiation is based on two principles: *i)* The component's code is always single-instance, and *ii)* the component's *context* is duplicated per new instance. By component context, we mean the *data structures* required to handle the properties independently from the component's code. Thus, a REMORA component becomes a *statically reconfigurable and reusable* entity and the memory overhead is kept very low by avoiding code duplication.

REMORA proposes three *multiplicity types* for the component' context: *raw-instance* (stateless component), *single-instance*, and *multiple-instances*. The REMORA engine features an algorithm determining the multiplicity type of a component based on: *i)* whether the component owns any property, *ii)* whether the component is an event producer, and *iii)* the number of components subscribed to a specific event. When the multiplicity type is determined, the REMORA engine statically allocates memory to each component instance.

2.3 Event Management

The REMORA design comprehensively supports event-based interactions between components. The event design principles in REMORA include:

Event Attributes. An event type in our approach can have a set of attributes with specific types. By defining attributes, the event producer can provide the event-specific information to the event consumer, *e.g.*, the `latency` attribute of TimerEvent in the Blink application.

Application Events vs OS Events. Events in REMORA are either *application-level events* or *OS-events*. Application events are generated by the REMORA framework (like Timer in the Blink application), while the latter are generated by OS. The REMORA runtime features mechanisms to observe OS-events, translate them into corresponding application-level events, and publish them through REMORA components.

Event Observation Interface. This interface is proposed to specify the time period during which events should be observed by producers, *e.g.*, the listening period of a TCP/IP event is the whole application lifespan (*automatic* observation), while a Timer event is observed according to the user-configured time (*manual* observation). REMORA proposes the *event observation interface* in order to control the manual observations. This generic interface includes operations, such as `start`, `pause`, `resume`, and `terminate`. If an event type is manually observable, the associated event producer should implement this interface. By doing that, the event consumer can handle the lifecycle of the observation process by calling operations in this interface without being aware of the associated event producer.

Event Configuration Interface. An event type can have an interface enabling the event consumer to configure event generation. Each component producing an event should implement the associated configuration interface identified in the specification of the event. This interface is designed to decouple completely the consumer and the producer.

Single Event Producer per Event Type. An event type in REMORA is produced by *one and only one* component. Instead of imposing the high overhead of defining event channels and binding manually event consumers and producers, the REMORA framework *autowires* producers and consumers. We believe that this constraint does not affect event-related requirements of applications. In case of having two producers generating one event type, we can define a new event type, extended from the original event, for one of the producers.

Event Casting. Events in our proposal can be either *unicast*, or *multicast*. Unicast is a one-to-one connection between an event producer and an event consumer (*e.g.*, TimerEvent), while a multicast event may be of interest to more than one component (*e.g.*, UserButtonEvent). The REMORA framework distinguishes between these two types in order to improve the efficiency of processing and distributing events. We also need to clear how multiplicity type of components on the one side, and unicast events and multicast events on the other side are related. To this end, we define two invariants:

Invariant1: *The consumer of a unicast event should be a raw-instance or single-instance component.*

Invariant2: *The producer of a multicast event should be a raw-instance or single-instance component.*

These invariants are mainly proposed to boost the efficiency of event processing in the REMORA framework. We do not support other event communication schemes since it implies to reify at runtime the source and the destination of an event and to maintain complex routing tables within the REMORA framework, which will induce significant overheads in term of memory footprints and execution time. We believe these invariants do not limit event-related logic of embedded applications.

Events Description. Similar to components, events have their own descriptions, which are in accordance to the event specification in REMORA, discussed above. Figure 6 presents a simplified events description document of the Blink application. This document consists of two outer tags: `event.remora` and `event.os`, corresponding to the application events and the OS-events, respectively.

```
<eventType>
 <event.remora type="core.sys.TimerEvent" observation="manual" castType="unicast">
  <attribute name="latency" type="xsd:int"/>
  <configInterface>
   <operation name="configure">
    <in name="interval" type="xsd:int"/>
    <in name="periodic" type="xsd:short"/>
   </operation>
  </configInterface>
 </event.remora>
 <event.os/>
</eventType>
```

Fig. 6. Application events description

2.3.1 Event Management Illustration

Figure 7 illustrates the event management mechanism implemented in REMORA. We explain the mechanism based on the steps labeled in the figure. During the first two steps, the event consumer can configure event generation and control event observation by calling the associated interfaces realized by the event producer component. These steps in our sample application are achieved in the Blink component (event consumer) by the code below:

```
aTimeEvent.producer.configure(3*CLOCK_SECOND, 1);
aTimeEvent.observation.start();
```

Note that the programmer is not aware of the TimerEvent producer. She/he only knows that the TimerEvent generator is expected to implement the `configure` function defined in the description of TimerEvent (cf. Figure 6). The TimerEvent producer should also implement the observation interface as the observation type of TimerEvent is manual.

Whereas the above steps are initiated by the programmer, the next two steps are performed by the REMORA framework. Step 3 is dedicated to *polling* the producer component to observe event occurrence. The event producer is polled by the REMORA framework through a *dispatcher* function in the producer. In fact, the event observation occurs in this function. The polling process is started, paused, resumed, and terminated based on the programmer's configuration for the event observation, performed in step 2.

For application-level events, the REMORA framework is in charge of calling periodically this function, while for OS-events, REMORA invokes this function whenever an OS-event is observed by the REMORA runtime. The REMORA runtime listens to only application-requested OS-events, and delivers the relevant ones to the framework. The REMORA framework then forwards the event to the corresponding OS-event producer component by calling its dispatcher function.

Finally, in step 4, upon detecting an event in the dispatcher function, the producer component creates the associated event, fills the required attributes, and publishes it to the REMORA framework. The framework in turn forwards the event to the interesting components by calling their event handler function.

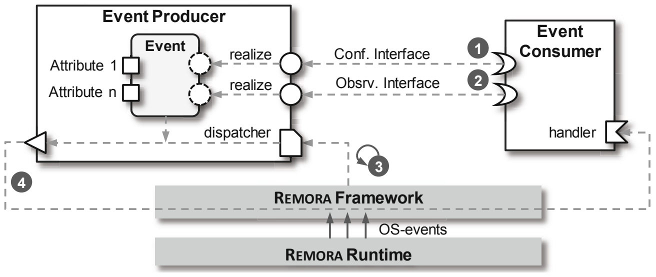


Fig. 7. Event management mechanism in REMORA

2.4 Components Assembly and Deployment

A typical WSN application may contain several implementations of a certain component type due to the existing heterogeneity in such platforms. To configure an application according to the target platform, REMORA introduces components *assembly* (equivalent to *composite* component in SCA). This XML document lists the application components, as well as bindings between their references and services. Figure 8 shows the configuration of Blink application in which there is only one binding from Blink to the Leds component implementing the `ILeds` interface for the MSP430 microcontroller. Note that the event-binding between Blink and Timer is created automatically by the REMORA framework.

Figure 9 demonstrates the four main phases of application deployment. The REMORA Development Box encompasses specification-supporting artifacts, as well as *External Types Definition*—a set of C header files containing application's type definitions. It should be noted that the component implementation can call OS libraries through a set of system APIs implemented by REMORA runtime components. Therefore, there is no hard-coded dependencies between REMORA implementers and the native API of the underlying OS. In the next phase, the REMORA engine reads the elements of

```
<composite name="app.BlinkAppConfigurer">
  <component name="ledControl">
    <implementation.remora implementer="cmu.telosb.peripheral.Leds"/>
  </component>
  <component name="blink">
    <implementation.remora implementer="app.BlinkApp"/>
  </component>
  <component name="timer">
    <implementation.remora implementer="core.sys.Timer"/>
  </component>
  <wire source="blink/iLeds" target="ledControl/iLeds"/>
</composite>
```

Fig. 8. Blink application configuration

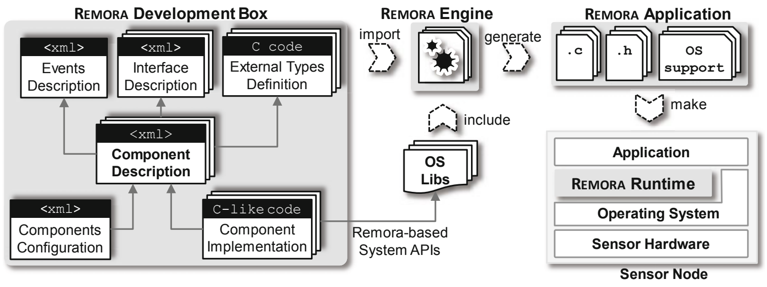


Fig. 9. REMORA-based development process

the development box and also OS libraries in order to generate the REMORA framework including the source code of components and OS-support code (for deployment). Then, application object file will be created through OS-provided facilities and finally deployed on sensor nodes.

3 Implementation

In this section, we discuss the key technologies, techniques, and methods used for the implementation of REMORA. We structure this section according to the phases proposed for REMORA-based application development.

3.1 REMORA Engine

The REMORA engine is designed to analyze the implementations of components and generate the equivalent C code, as well as OS-support code. The engine is written in Java because of its cross-platform capabilities, as well as its strong support for XML processing. Additionally, the object-oriented nature of Java simplifies the complex process of code analyzing and code generation. We briefly discuss the key design issues of the engine below.

The first concern of the REMORA engine is the mechanism for parsing the C-like implementation of components. To this end, we have developed a parser module, which is originally generated by ANTLR—a widely used open-source parser generator [13]. We have modified the generated parser to extract REMORA-required information, such as name, signature, and body of implementation functions.

Dealing with events, component instantiation and component configuration is the other key part of the REMORA engine. This unit deduces the multiplicity type of components and generates the necessary data structures. It also features a set of well-defined techniques, such as *in-component call graph analyzer* and *cross-component call tracker* to support stateful components. The former concept is concerned with discovering context-dependent functions of a component, and the latter tracks the interactions between components in order to retain the state of components. Finally, the major task of this part is to embed framework-support patches in the component implementation.

3.2 REMORA Framework

The REMORA framework is mainly designed to facilitate event management tasks, including *scheduling* and *dispatching*. To explain these tasks, we first introduce two *queue* data structures supporting our event model. The first queue is dedicated to the event producer components (PQ), while the second one is designed to maintain the event consumers (CQ). We discuss here how the REMORA framework is built based on these data structures.

Scheduling in REMORA refers to all arrangements required to *enqueue* and *dequeue* event producers and event consumers. In particular, the main concern is *when* to enqueue/dequeue a component and *who* should perform these tasks. The REMORA framework addresses these issues based on the observation model of events. For example, if an event is *automatically* observable, the associated producer component and all the subscribed consumers are enqueued by the framework core during the application startup, while in a *manual* observation, producer and consumer are placed respectively in PQ and CQ when the consumer component calls the `start` function of observation interface.

Figure 10 illustrates the *dispatching* mechanism in the framework including the supporting data structures. In *Polling*, the REMORA framework continuously polls the EventProducer components through *dispatcher*—the globally known callback function. Whenever a producer dispatches an event (AbstEvent), the framework casts this event to the actual event type, which is either UCastEvent(unicast event) or MCastEvent(multicast event). UCastEvent will be directly forwarded to the subscribed consumer through the callback function pointer stored in the UCastEvent. If a MCastEvent is generated, the framework delivers it to all the interesting components formerly enqueued. For OS-events, the same procedure is followed except the polling phase, which is performed by the operating system.

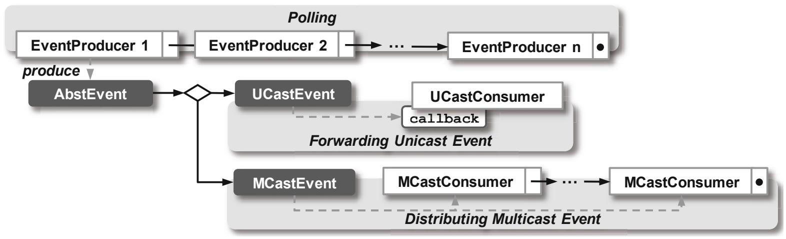


Fig. 10. REMORA event processing mechanism

3.3 REMORA Runtime

The current implementation of the REMORA runtime is a Contiki-compliant *process* running together with all other *autostart* processes of Contiki. This process undertakes two tasks: *i)* periodically scheduling the REMORA framework (for polling event generator components) to run, and *ii)* listening to the OS-events and delivering the relevant ones to the REMORA framework. By relevant, we mean the REMORA runtime recognizes those OS-events that are of interest to the application. To achieve such filtering, the source code of this part is generated by the REMORA engine according to the events description (cf. Section 2.3) of target application and then imported to the REMORA runtime. By doing that, we provide a lightweight event distribution mechanism interpreting only application-specific OS-events.

Additionally, the application code may need to use OS-provided libraries. REMORA proposes system API *wrapper* components for this purpose. In fact, these components delegate all high-level system calls to the corresponding OS-level functions, *e.g.*, the `currentTime()` function call in the system API is delegated to the Contiki function `clock_time()`. We offer this API to fully decouple the application components from OS modules and ensure the portability of REMORA. If an application is not expected to be ported to other platform types, the OS libraries can be directly called within the component implementation.

4 Evaluation

In this section, we first demonstrate and assess a real REMORA-based application, then we focus on the general performance figures of REMORA.

4.1 A Real REMORA-Based Deployment

Our real application scenario is a network-level *application suite* consisting of a set of mini applications bundled together. This suite is basically designed to provide services, such as *code propagator* and *web facilities* in WSNs. We focus here on the first one and design it based on the REMORA approach.

Code propagation becomes a very important need in WSNs when we need to update remotely the running application's software [27]. The code propagator application is responsible for receiving all segments of a running application's object code over the network and loading the new application image afterwards. The code propagator exploits the TCP and UDP protocols to propagate code over the network. At first, TCP is used to transfer new code, block by block, to the sink node connected to the code repository machine, and then UDP is used to broadcast wirelessly new code from a sink node to other sensor nodes in the network. When all blocks are received, the code propagator loads the new application.

Figure 11 shows the components involved in the first part of our application scenario. TCPListener is a core component listening to TCP events. This multiple-instances event generator is created for each TCP event consumer component with unique listening port number. For example, CodePropagator receives data from port 6510 (`codePropPort`), while WebListener is notified for all TCPEvents on port 80 (`webPort`). CodePropagator stores all blocks of new code in the external flash memory through the `IFile` interface implemented by the FileSystem component. When all blocks are received, CodePropagator loads the new application by calling the `ILoader` interface from the ELFLoader component. These two interfaces are system APIs that delegate all application-level requests to the OS-specific libraries. The `INet` interface, implemented by the Network component, is also the other system API providing the low-level network primitives to TCPListener.

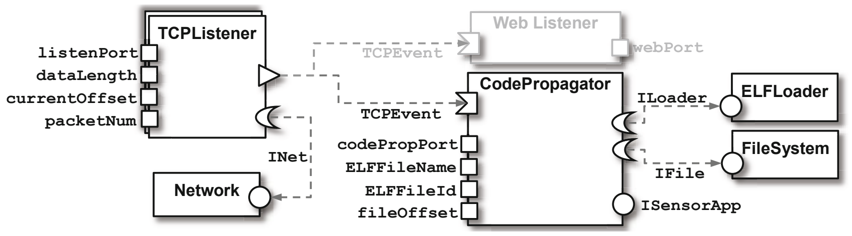


Fig. 11. Code propagation application architecture

As mentioned before, we adopt Contiki as our OS platform to assess the REMORA component model. Contiki is being increasingly used in both academia and industrial applications in a wide range of sensor node types. Additionally, Contiki is written in the standard C language and hence REMORA can be easily ported to this platform. Finally, the great support of Contiki on event processing and process management motivate us to design and implement the REMORA runtime on this OS. Our hardware platform is the popular TelosB mote equipped with a 16-bit TI MSP430 MCU with $48KB$ ROM and $10KB$ RAM.

The concrete separation of concerns in this application is the first visible advantage of using REMORA. The second improvement is the *easy* reuse of TCPListener for other TCP-required applications, which is not the case in a non-componentized implementation. In particular, for each new application, we only need to instantiate the *context* of TCPListener and configure its properties (like port number) accordingly, *e.g.*, WebListener in Figure 11.

Table 1 reports the memory requirement of REMORA and Contiki programming model (*protothreads*) for implementing the code propagation application. As indicated in the table, the REMORA-based development does not impose additional data memory overhead, while it consumes extra 532 bytes of code memory, which is essentially related to the cost of framework and runtime modules. This cost is paid once and for all, regardless of the size and the number of applications running on the sensor node. The code memory cost can be even further reduced by removing system APIs (Network, FileSystem, and ELFLoader) and calling directly the Contiki's libraries within CodePropagator. Note that the overhead of TCPListener can also be decreased when this component is shared for the use of other applications, *e.g.*, WebListener. Therefore, we can conclude that the memory overhead of REMORA is negligible compared to the high-level features it provides to the end-user.

The rest of this section is devoted to the assessment of two main performance figures of REMORA, namely, memory footprints and CPU usage.

Table 1. The memory requirement of code propagation application in REMORA-based and Contiki-based implementations

Programming Model		Code Memory (bytes)	Data Memory (bytes)
Contiki		**722**	**72**
REMORA	Code Propagation Components		
	CodePropagator	252	36
	TCPListener	310	0
	System API Components		
	ELFLoader	38	0
	Network	92	0
	FileSystem	68	0
	REMORA Core		
	Framework and Runtime	494	14
	Total	**1254**	**50**
REMORA overhead		**+532**	**-22**

4.2 Memory Footprint

High memory usage has been one of the main reasons behind unsuccessfulness of component-based proposals for embedded systems. In REMORA, we have made a great effort to maintain memory costs as low as possible. The first step of this effort is to avoid creating meta-data structures, which are not beneficial in a static deployment. Distinguishing unicast events and multicast events has also led to a significant reduction in memory footprints as REMORA does not need to create any supporting data structure for unicast events.

The memory footprints in REMORA is categorized into a minimum overhead and a dynamic overhead. The former is paid once and for all, regardless of the amount of memory is needed for the application components, while the latter depends on the size of application. Table 2 shows the minimum memory requirements of REMORA, which turn out to be quite reasonable with respect to both code and data memory. As mentioned before, our sensor node, TelosB, is equipped with $48KB$ of program memory and $10KB$ of data memory. As Contiki consumes roughly $24KB$ (without µIP support) of both these memories, REMORA has a very low memory overhead considering the provided facilities and the remaining space in the memory.

Table 3 shows the memory requirement of different types of modules in the REMORA framework. The exact memory overhead of REMORA depends on how an application is configured, *e.g.*, an application, containing one single instance event producer and one unicast event, needs extra 56 bytes (38+8+10) of both data and code memory. Ordinary components do not impose any memory overhead as REMORA does not create any meta data structures for them. For other types of modules, REMORA keeps the data memory overheads very low as this memory in our platform is really scarce. We also believe that the code memory overhead is not significant since a typical WSN application is small in size and it may contain up to a few tens of components, including ordinary

Table 2. The minimum memory requirement of REMORA

Module	Code Memory (bytes)	Data Memory (bytes)
Framework Core	374	4
Runtime Core	120	10
Total	**494**	**14**

Table 3. The memory requirement of different entities in REMORA

Entity		Code Memory (bytes)	Data Memory (bytes)
Ordinary Component		0	0
Event Producer	Single Ins.	38	8
	Multiple Ins.	42	10
Event	Unicast	0	10
	Multicast	0	10
Multicast Event Consumer		30	6
OS Event		28	4
System API		4	0

components. It should be noted that componentization itself reduces the memory usage by maximizing the reusability degree of system functionalities like the one discussed in the code propagation application.

4.3 CPU Usage

As energy cost of REMORA core is limited to only the use of the processing unit, we focus on the processing cost of our approach and show that REMORA keeps the CPU usage at a reasonable level, and in some configurations it even reduces CPU usage compared to the Contiki-based application development.

To perform the evaluation, we set up a Blink application in which a varying number of mirror components (1 to 15) switch LEDs on and off every second. The two implementations of this application, Contiki-based and REMORA-based, were compared according to a CPU measurement metric. The metric was to measure the amount of time required by one REMORA component and one Contiki process to switch LEDs six times: three times on and three times off. With the less number of switches, we cannot extract the exact timing differences as our hardware platform provides a timing accuracy of the order of one millisecond.

We started our evaluation by deploying an application like the one presented in Section 2.1 and measuring the CPU usage based on our metric. In each next evaluation step, we added a mirror Blink component to the application and measured again the time. This experiment was continued for 15 times. We made the same measurement for a Contiki-based Blink application and added a new Contiki Blink process in each step. Figure 12 shows the evaluation result of our scenario. When we have one Blink component/process, the CPU overhead of both approaches is almost the same, indicating that the REMORA runtime and framework impose no additional processing overhead. When the number of components/process increases towards 15, reduction in CPU usage is achieved in two dimensions.

Firstly, the number of CPU cycles for REMORA is slightly less than for the Contiki application. This difference reaches 13 milliseconds when Contiki undertakes running 15 Blink processes. Therefore, we can conclude that REMORA does not impose additional processing overhead affecting the performance of the system. Secondly, the CPU usage of REMORA application is reduced when the number of Blink components is increased. This improvement is achieved because the number of context switches between the REMORA runtime and the REMORA framework is significantly decreased when there are more event producer components (Timer) in PQ.

To clarify this issue, we assume that the application running time is T and Contiki periodically allocates CPU to the REMORA runtime in this period. In each allocation round, the runtime module invokes the event manager in the REMORA framework to poll the application level event producers. Given that there are K producers in PQ, the polling process consumes $K \times t_1$ of CPU, where t_1 is the average processing cost of one element. Therefore, the frequency of event manager calling (equal to the number of context-switches) is in the order of $T/K \times t_1$. Therefore, as the value of K is increased the number of context-switches is decreased accordingly. Figure 13 shows the changes in the number of context-switches when the number of Timer components is increased to 15. As a result, the maximum performance in REMORA relies on the average number of event producer components enqueued during the application lifespan, while in the worst case (a very few producers in the queue) REMORA does not impose any additional processing cost.

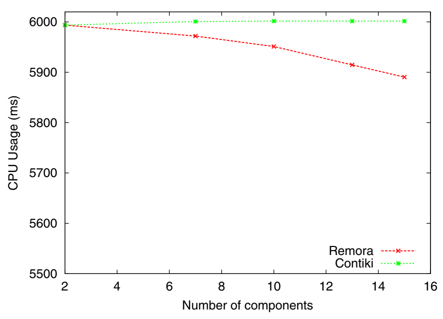


Fig. 12. The REMORA-based implementation does not impose additional CPU overhead compared to the Contiki-based implementation

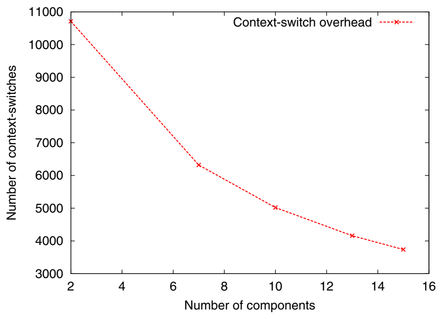


Fig. 13. As the number of producer components in the queue is increased, the number of context switches is significantly decreased

5 Existing Approaches

In this section, we survey the existing component-based approaches for node-level programming on embedded system and WSNs. Most of these component models mainly aim at building entire operating systems as an assembly of components.

In the area of WSNs, NESC [8] is perhaps the best known component model being used to develop TINYOS [9]. As mentioned earlier, the main downside of NESC is that it is tightly bound to the TINYOS platform. Moreover, although NESC efficiently supports event-driven programming, events in NESC are not considered as independent entities with their own attributes and specifications. Therefore, the binding model of event-related components is not well-described as it is not essentially described based on the specification of events. Additionally, the unique features of REMORA, such as multiplicity in component instance and property-based reconfiguration of components bring significant improvements to component-based programming in WSNs compared to NESC.

Coulson et al. in [14] propose OPENCOM as a generic component-based programming model for building system applications without dependency on any target-specific platform environment. The authors express that they have tried to build OPENCOM with negligible overhead for supporting features specific to a development area, however it is a generic model and basically developed for platforms without resource constraints and tends to be complex for embedded systems. To evaluate OPENCOM, we deployed a sample *beacon* application [15], including Radio, Timer and Beacon components, on a TelosB node with Contiki. Based on our measurements, the memory footprint of this application is significantly high, so that it consumes $4,618$ bytes of code memory and 28 bytes of data memory.

The OSGi model [16] is a framework targeting powerful embedded devices, such as mobile phones and network gateways along with enterprise computers. OSGi features a secure execution environment, support for runtime reconfiguration, lifecycle management, and various system services. While OSGi is suitable for powerful embedded devices, the smallest implementation, Concierge [17] consumes more than $80KB$ of memory, making it inappropriate for resource-constrained platforms.

OSKIT [18] is a set of ready-made components for building operating systems. OSKIT is developed with a language called KNIT [19]. In contrast to NESC, KNIT is not limited to OSKIT. OSKit has adapted the Microsoft COM model and is not primarily focused on embedded systems.

The THINK framework [20] is an implementation of the FRACTAL [21] component model applied to operating systems. The choice of the THINK framework is motivated by the fact that it allows fine-grained reconfiguration of components. Although the experiments on deploying THINK components on WSNs have been quite promising in terms of memory usage [23], the lack of application-level event support is the main hurdle for using THINK in WSNs. LOOCI [24] is another component-based approach, providing a loosely-coupled component infrastructure focusing on an event-based binding model for WSNs. However, the Java-based implementation of LOOCI limits its usage to the SunSPOT sensor node.

6 Discussion, Conclusion and Future Direction

We presented REMORA, a novel programming abstraction for resource-constrained embedded systems. The main motivation behind proposing REMORA is to simplify high-level event-driven programming in WSNs by a component-based approach. Moreover, involving PC-based developers in WSN programming and considering the state-of-the-art technologies for component development are two other challenges addressed by REMORA. The special consideration paid to the event abstraction in REMORA makes it a practical and efficient approach for WSN applications development. The other key features of REMORA include: applicability on a wide range of embedded OSs, rich support of component reusability and instantiation, and reduced effort and resource usage in WSN programming.

Careful restrictions on the REMORA component model, including the lack of dynamic memory allocation and avoiding M-to-N communications between event producers and event consumers bring significant improvements to the static deployments in WSNs. Since one of our main future directions is to support dynamic component reconfiguration in REMORA [25, 26, 27], we encounter a new major challenge on how to efficiently provide such a feature in REMORA so that the overhead of dynamic memory allocation is carefully minimized.

As mentioned earlier, the current goal of REMORA is to be exploited only in application-level programming. However, we believe that the efficient support of event processing in REMORA potentially enables it to componentize system level functionalities. In the Blink application, we implicitly demonstrated this capability by redeveloping the Timer component, which is essentially developed at the OS level. To address precisely this issue, we need to enhance the current REMORA implementation with features like *concurrency support*, *task scheduling*, and *interrupts handling*.

In our current implementation, a REMORA process cannot be preempted by any other process in the operating system. This issue becomes critical when a component execution takes a long time to complete and it causes large average waiting times for other processes waiting for the CPU. The event handling model of REMORA can be used to provide preemption by defining a new event type per preemption-required point of application, while in this case the component implementation and the event management become quite complicated. This concern will also be considered in the future extensions for REMORA. In particular, we intend to promote the native Contiki macros, handling process lifecycle, to the REMORA application level. In this way, the REMORA component becomes preemptable by explicitly yielding the running process.

Beside the fact that REMORA provides a strong abstraction for single node programming, the same level of programming abstraction is expected to occur at the network level. This challenge opens up another key area for future work: how to make REMORA components distributed by the provision of a well-defined remote invocation mechanism.

Acknowledgments. This work was partly funded by the Research Council of Norway through the project SWISNET, grant number 176151.

References

1. Sugihara, R., Gupta, R.K.: Programming models for sensor networks: A survey. ACM. Trans. Sensor Networks 4(2), 1–29 (2008)
2. Szyperski, C.: Component Software: Beyond Object-Oriented Programming, 2nd edn. ACM Press and Addison-Wesley, New York (2002)
3. Bachmann, F.L., et al.: Technical Concepts of Component-Based Software Engineering, 2nd edn. Carnegie Mellon Software Engineering Institute (2000)
4. Ommering, R., Linden, F., Kramer, J., Magee, J.: The Koala component model for consumer electronics software. IEEE Computer 33(3) (2000)
5. Winter, M., et al.: Components for embedded software: the PECOS approach. In: Proc. of the CASES 2002. ACM Press, New York (2002)

6. Hansson, H., Akerholm, M., Crnkovic, I., Torngren, M.: SaveCCM-a component model for safety-critical real-time systems. In: Proc. of the IEEE Euromicro Conference (2004)
7. Plsek, A., Loiret, F., Merle, P., Seinturier, L.: A Component Framework for Java-Based Real-Time Embedded Systems. In: Issarny, V., Schantz, R. (eds.) Middleware 2008. LNCS, vol. 5346, pp. 124–143. Springer, Heidelberg (2008)
8. Gay, D., et al.: The nesC Language: A Holistic Approach to Networked Embedded Systems. In: Proc. of the SIGPLAN Conference on Prog. Language Design and Impl. (2003)
9. Levis, P., et al.: TinyOS: An Operating System for Sensor Networks. Ambient Intelligence (2005)
10. http://www.oasis-opencsa.org/sca
11. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proc. of 1st Workshp. on Embedded Networked Sensors (2004)
12. Dunkels, A., Schmidt, O., Voigt, T., Ali, M.: Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In: Proc. ACM SenSys (2006)
13. ANTLR, http://www.antlr.org
14. Coulson, G., et al.: A generic component model for building systems software. ACM Trans. Computer Systems, 1–42 (2008)
15. Wisebed, http://www.wisebed.eu/wiki/pmwiki.php?n=Main.Osaapp1
16. The OSGi Alliance. The OSGi framework (1999), http://www.osgi.org
17. Rellermeyer, J., Alonso, G.: Concierge: A Service Platform for Resource-Constrained Devices. ACM SIGOPS Operating Systems Review 41(3), 245–258 (2007)
18. Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., Shivers, O.: The Flux OSKit: A Substrate for Kernel and Language Research. Operating Systems Principles (1997)
19. Reid, A., Flatt, M., Stoller, L., Lepreau, J., Eide, E.: Knit: Component Composition for Systems Software. In: Operating Systems Design and Implementation (OSDI) (2000)
20. Fassino, J.-P., Stefani, J.-B., Lawall, J., Muller, G.: Think: A software framework for component-based operating system kernels. In: Proc. of the USENIX Annual Conference (2002)
21. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The Fractal component model and its support in Java. Softw., Pract. Exper. (2006)
22. Remora. Website, http://folk.uio.no/amirhost/remora
23. Lobry, O., Navas, J., Babau, J.: Optimizing Component-Based Embedded Software. In: 2nd IEEE Workshop on Component-Based Design of Resource-Constrained Sys., COMPSAC 2009 (2009)
24. Hughes, D., et al.: LooCI: A loosely-coupled component infrastructure for networked embedded systems. Mobile computing Multimedia (2009)
25. Taherkordi, A., et al.: WiSeKit: A Distributed Middleware to Support Application-Level Adaptation in Sensor Networks. In: Senivongse, T., Oliveira, R. (eds.) DAIS 2009. LNCS, vol. 5523, pp. 44–58. Springer, Heidelberg (2009)
26. Taherkordi, A., Rouvoy, R., Le-Trung, Q., Eliassen, F.: A Self-Adaptive Context Processing Framework for Wireless Sensor Networks. In: Proc. of ACM MidSens 2008, Belgium (2008)
27. Mottola, L., et al.: Selective Reprogramming of Mobile Sensor Networks through Social Community Detection. In: Proc. of EWSN 2010, Portugal (2010)

Stateful Mobile Modules for Sensor Networks

Moritz Strübe, Rüdiger Kapitza, Klaus Stengel, Michael Daum, and Falko Dressler

Dept. of Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, Germany
{struebe,rrkapitz,stengel,daum,dressler}@cs.fau.de

Abstract. Most sensor network applications are dominated by the acquisition of sensor values. Due to energy limitations and high energy costs of communication, in-network processing has been proposed as a means to reduce data transfers. As application demands may change over time and nodes run low on energy, get overloaded, or simply face debasing communication capabilities, runtime adaptation is required. In either case, it is useful to be able to migrate computations between neighboring nodes without losing runtime state that might be costly or even impossible to recompute. We propose *stateful mobile modules* as a basic infrastructure building block to improve adaptiveness and robustness of in-network processing applications. Stateful mobile modules are binary modules linked on the node itself. Even more importantly, they can be transparently migrated from one node to another, thereby keeping statically as well as dynamically allocated memory. This is achieved by an optimized binary format, a memory-efficient linking process and an advanced programming support.

1 Introduction

A large fraction of Wireless Sensor Network (WSN) applications target long-term monitoring of environmental conditions. Typical examples are monitoring of trees, volcanoes, glaciers, and buildings [1]. All these applications collect various sensor values and transport them to more powerful gateway nodes at the edge of the sensor network. Energy is commonly the limiting factor of long-term monitoring experiments in the context of WSNs. Therefore, reducing communication, which is one of the most energy-intensive tasks in this domain, is crucial. *In-network processing*, the pre-processing of sensor data inside the network is a powerful technique to significantly reduce the amount of data to be transferred [2, 3]. However, in many scenarios, the optimal pre-processing has to be determined at runtime. Furthermore, nodes in this domain can run low on energy, get overloaded, or face worsening network conditions. In all these cases, the relocation of pre-processing operators is a basic building block to continue service provisioning. The demand to keep application state is especially challenging in the context of in-network processing, despite relocation. Otherwise, the result is data loss, which can cause blind spots in monitoring experiments decreasing the overall data quality and in the worst case losing important events thereby rendering them useless. Even if it is possible to replace lost data, this can take a considerable amount of time and resources, e.g., using a *pause-drain-resume* strategy [4].

In order to address the aforementioned demands for adaptability and to minimize data loss, resource-efficient system support has to be provided that enables the dynamic deployment and migration of applications in a state preserving manner. However, even the most basic task to fulfill the goal of dynamic stateful migration, the software deployment, is cumbersome in sensor networks. Code has to be transferred to target nodes, requiring non-negligible communication and energy efforts. As a consequence, several research activities targeted to provide mechanisms and infrastructures to efficiently deploy software in WSNs at the level of system images [5], modules (prelinked or linked at runtime) [6,7,8], and byte code [9,10]. Only, a limited number of systems thereby preserve the execution state of updated code [11,7], and they all fall short on migration support. Thus, application developers have to do this manually by providing custom serialization routines [12] or rely on a high level byte code language, which has the drawback of a resource-intensive interpretation and a considerable overhead due to the required runtime environment [13].

Taking these facts into account, we propose the concept of *stateful mobile modules*. It enables dynamic migration of stateful, native modules inside a WSN. This is achieved by combining a set of techniques starting with a size-optimized binary format and a memory-efficient linking process. The latter provides the freedom to deploy the same native code on multiple nodes and migrate code inside a WSN as needed. To enable transparent migration of in-memory module state, we provide a programming model similar to high-level languages, such as Java and C#, for supporting serialization. All relevant statically allocated variables that have to survive a migration are marked in the source code. For dynamically allocated memory and pointer variables therein, additional actions are needed. Here, we use a smart-pointer approach provided by an easy to use API. We implemented *stateful mobile modules* and the associated programming model as a resource-efficient layer on top of the Contiki Operating System [14] and evaluated its benefits in a realistic in-network processing scenario.

In the remainder of the paper, we first outline an introductory application scenario. Sections 3 and 4 introduce our system support for resource-efficient linking and for runtime migration. Next, we detail evaluation results (Section 5) and briefly summarize related approaches (Section 6). Finally, Section 7 concludes the paper.

2 Overview

In the following, we outline a data stream processing example, which represents a typical use case for stateful mobile modules. Next, we summarize the derived goals that we took into account for building the proposed system support.

2.1 Environmental Monitoring Stream Processing Example

Fig. 1 depicts a distributed stream processing query targeting the long-term monitoring of microclimate changes on a rock. The query is composed of a set of connected stream operators distributed over seven nodes: one taking the role of a gateway to the sensor network and six additional nodes that build the actual WSN. Besides receiving data from the network, a server connected to the gateway controls the placement and the wiring of the stream processing operators. These operators are structured as stateful mobile modules so they can be dynamically distributed.

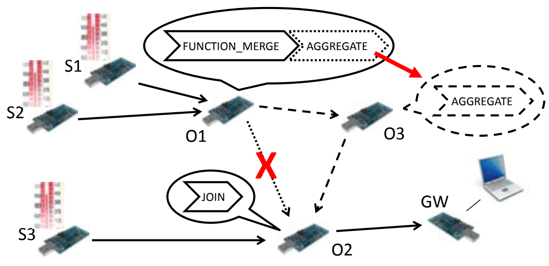


Fig. 1. Migrating of the AGGREGATE operator due to system resource shortage at its current node

In our example scenario, the outer left nodes create three streams each providing temperature data: S1, S2, and S3. The distributed query creates outputs values if the temperature of S3 provided by a sensor placed near the ground is lower than in the area of S1 and S2, both located on top of the rock. A JOIN operator (O1) delivers these items to a sensor node that is connected to the gateway (GW). In our scenario the values of S1 and S2 might be erroneous due to isolation (e.g., only one sensor is exposed to direct sunlight), so we implemented a simple way of sensor data cleaning by using a minimum function provided by FUNCTION_MERGE and an AGGREGATE operator for smoothing outliers (e.g., caused by clouds). Both are placed on an intermediate node on the stream path (O1).

In the following, the migration of the AGGREGATE operator instance is described. It calculates the mean of a configurable number of samples. For example, due to energy reasons and worsening communication, the central server decides to integrate a neighboring node (O3) into the distributed query by initiating the migration of this operator. This includes transferring the module and its state, and rerouting the data flow. Furthermore, the new node has to receive the results of FUNCTION_MERGE and to deliver the results to the host of the JOIN operator. Other scenarios might include the migration of FUNCTION_MERGE or relocation of the JOIN operator. In all these cases, stateful mobile modules enable a code and run-time state migration that is transparent from an operator's point of view.

2.2 Goals

From the described scenario and targeted more complex ones [15], we derived the following goals and requirements for providing stateful mobile modules:

Distribution of modules. Modules generated on a host outside the WSN can be sent to one or more sensor nodes. Furthermore, module code can be shared among nodes by direct exchange. Modules should only require minimal runtime support besides the Operating System (OS) and need to be provided in a space-efficient format. The former avoids overhead during execution, e.g., opposed to a byte-code-based approach, the latter targets low communication costs.

Linking, loading, and running of modules. Consequently, modules should be linked on the node. This allows the use of the same module on nodes with slightly different kernels, e.g., different minor versions or supported hardware. Additionally, the linking process should be memory-efficient and fast. The former leaves more space for

applications the latter enables faster integration and therefore implicitly aims at reducing energy demand.
Migration of modules. It should be possible to migrate a module to a new node with minimal disruption, thereby preserving its execution state. This means that in-memory data, including static variables as well as dynamically allocated memory, is automatically copied to the new node and can be utilized right away. The rationale behind this requirement is not to lose costly computed state information, e.g., gained by long-term monitoring, to relief application developers from the burden to provide custom operations to preserving data, and, lastly, to keep services permanently available.

3 System Support for Mobile Modules

In the following, we detail our support for resource-efficient distribution of native modules using our custom object format (*Minilink*) and a memory-optimized node-level linking process.

3.1 Background: Linking and Loading in WSNs

When compiling a C/C++ file into an object file, the compiler translates the source code into binary code whereas the linker is responsible for the actual memory layout making the code ready for execution. Thus, the compiler writes placeholders into the code and adds an entry for each variable and function to the *relocation table*. Further on, all functions and variables that might be externally accessed are added to the *symbol table*. Next, the linker uses the symbol tables and the target memory location of the code to resolve all references in the relocation table for substituting the placeholders.

For adding a new module to a sensor node, it must be placed in memory and linked against the functions provided by the kernel. If both the modules and the kernel are known in advance, this can be performed at a different machine outside the WSN (*pre-linking*) [7]. However, even slight differences, such as the use of different compiler or linker versions, can cause incompatible modules. Additionally, the placement of modules is fixed, which can lead to collisions due to the limited available memory if further modules are added over time.

Alternatively, one can link on the node itself (*runtime-linking*), which has been proven an effective way of distributing code in a network with slightly heterogeneous kernels [8]. Dunkels et al. implemented a linker for the Executable and Linking Format (ELF) format, which they identified as too resource consuming for sensor nodes (see also Section 5). For this reason, they increased the efficiency by introducing Compact ELF (CELF), a custom ELF inspired binary format that is tailored to a 16-bit address space instead of 32 bit. However, a small code size is only one aspect that has to be taken care of, because the symbol table of a typical sensor OS kernel is several kilobytes and the linking process requires random access to the symbol table and the linked module. Whereas the first aspect substantially reduces the available memory, the second leads to a time and energy consuming linking process. This is even worsened by the fact that an ELF binary is subdivided into multiple sections, e.g., different program section like code (`.text`) and variables (`.data`). This design was made for flexibility and is not suitable for resource-restricted systems that rely on flash memory. First, flash can typically only be modified at the granularity of segments and, secondly, it is usually not possible to buffer the whole program section of a module in RAM. Due to the use of multiple sections, this causes a lot of costly random access during the linking process of a module that usually has to reside on a slow external flash.

We address both problematic aspects of runtime-linking by an optimized symbol table and a further compacted binary format. In combination, this enables an efficient linking process building the essential basis for supporting dynamic migratable modules.

3.2 Resource-Efficient Linking Using Minilink

In the following, we describe the different aspects of Minilink.

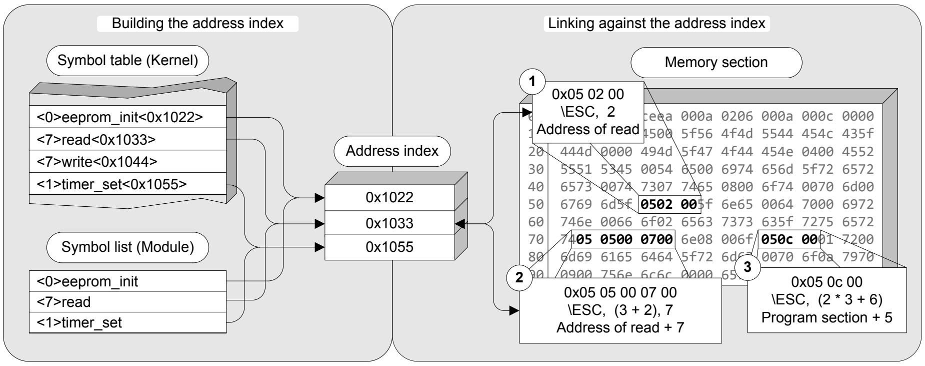
Placement of the Symbol Table. The symbol table of our implementation basis, the Contiki OS kernel, occupies about 5 to 6 KB of memory. This is rather large compared to the 48 KB internal flash of the TelosB[1] node, the platform we used for evaluations. However, most sensor nodes are equipped with external storage. The TelosB platform, for example, has an external flash of 1 MB, dedicated to store data. As the symbol table is only accessed during the process of linking, it can be placed on the external flash. The latter saves valuable internal memory for running applications.

Optimizing the Symbol Table. Many functions provided by the kernel have a common name prefix to indicate their relation to a module, e.g., `eeprom_read` and `eeprom_write`. As a consequence, we order the symbol table alphabetically and, instead of repeating a matching prefix, we store the size of the common prefix and the remainder of the function name. The symbol table in Fig. 2 shows an example for the `eeprom` function set. Each of the three functions starts with `eeprom_`. The first entry shares no characters with its previous entry, the following two the leading 7. Thus, 11 byte can be saved (14 saved by compression, 3 lost for indicating the prefix size).

In addition, we take advantage of the fact that symbol names are encoded in the 7-bit ASCII character set [16] by using the unused last bit to terminate them. This saves one byte per table entry. Finally, we exploit that symbols are sorted by name and, therefore, symbols of the same module are in consecutive order. Accordingly, they are also co-located within the code and can, under some checked precondition, be addressed relative to the previous symbol. This demands for only one byte instead of two bytes for the absolute address.

Stream-based Sequential Linking. Due the outlined memory-intensive linking process of ELF binaries, we propose a *stream-based sequential linking* approach, meaning that each byte needs only to be read once. Our linking process works in two stages: First, an *address index* is built. Secondly, the different sections are linked to their destination in a sequential order. To achieve this, our Minilink format structures modules in three sections: A header containing general information about the module, an alphabetically sorted and compressed list of used symbols, and the binary data itself. In contrast to the ELF file format, we do not have a relocation table as the relocation entries are directly woven into the binary data. The header contains only information that is essentially needed to link the module: the size of each section, the module name,

[1] The Xbow TelosB was formerly sold by Moteiv under the name Tmote Sky.



```
          ceea 000a 0206 000a 000c 0000
          4500 5f56 4f4d 5544 454c 435f
20   444d 0000 494d 5f47 4f44 454e 0400 4552
30   5551 5345 0054 6500 6974 656d 5f72 6572
40   6573 0074 7307 7465 0800 6f74 0070 6d00
50   6769 6d5f 0502 005f 6e65 0064 7000 6972
60   746e 0066 6f02 6563 7373 635f 7275 6572
70   7405 0500 0700 6e08 006f 050c 0001 7200
80   6d69 6165 6464 5f72 6d6  0070 6f0a 7970
90   0900 756e 6c6c 0000 65
```

Fig. 2. Mapping used for relocation

the entry point, and the number of required external symbols. Based on this information, the linker ensures that the node offers enough memory to link the module.

Building the Symbol Index. A module starts with a list of symbols required for linking. They are saved applying the same concepts as used for compacting the symbol table but miss address information. The latter is provided by matching the module symbols against the symbol table. As a result, we get the address index that contains all resolved symbol addresses of a module and builds the input for the linking phase (see Fig. 2). As both lists are ordered alphabetically, they can be processed in a sequential order. Together with the module header, the address index is the only data that has to reside in RAM during this process.

Linking Using the Symbol Index Using Minilink, all relocation entries are directly woven into the binary data. To identify them, these two bytes large entries are marked by a preceding escape sequence. We chose the escape sequence to be `h05` as this is neither a binary command on the TelosB platform nor is the escape-character itself commonly used in ASCII strings. Still `h05` can be encoded by `h050000`. The access to the address index starts with one, as zero is already reserved.

Fig. 2, (1) shows the escape sequence for the second symbol. Accordingly, the sequence `h050200` is replaced by the second value of the address index `h3310` that further on is written to memory. Sometimes, not only a symbol is referenced by a relocation entry, but additionally an offset is added, e.g., when accessing an element of a struct or array. For these cases, we reference the address index with an offset, and this time the value of the word following is added (2). Higher escaped values map to the different sections of the module containing the program code or variables (3).

4 Stateful Migration

In the context of in-network data processing, the migration of state in terms of statically and dynamically allocated variables is important as these often capture long-term execution results. In contrast, the execution stack representing the call history is of minor importance as applications in this domain are rather small and moderately complex. Furthermore, transferring the stack would introduce additional costs in terms of data transfer. Therefore, we support *weak migration* [17], meaning only application state is transferred but no execution-dependent state, such as values on the stack and CPU registers. This is also in line with the lightweight thread model of Contiki. Here, so-called Protothreads [18] lose their stack and register values when yielding the CPU in favor of another thread. However, once a Protothread is resumed, it still continues execution at the same position the CPU was released. Accordingly, we keep this behavior by restarting an application at the same point where it was suspended before migration. Thus, from an application programmer's perspective, releasing the CPU for another application and migrating to a different node are equivalent: in both cases, the stack is lost and the execution is continued immediately after the last executed statement.

In contrast to the execution stack, the handling of pointers still needs special attention when migrating modules using Protothreads. To keep them valid despite migration, one option would be to place data at the same memory address. Due to the limited available resources and the use of multiple modules as well as the absence of a Memory Management Unit, this is not practical for sensor nodes. However, if variables are dynamically allocated, there is no way to avoid the use of pointers. Therefore, mechanisms must be provided to properly access the data despite relocation. If the placement of pointers is known, they can be adjusted to the new memory layout. This reduces the burden for the developer to the necessary minimum.

4.1 General Process and Programming Model

In general, it is not reasonable to migrate all statically and dynamically allocated variables. If values can be easily recomputed or are dispensable to provide a service, they should be excluded from migration. We therefore provide two macros (`MIGRATABLE` and `MIGRATABLE_POINTER`), which assign a *section attribute* to the variable. This attribute instructs the compiler to put a variable in a special memory section. These extra memory sections are supported by our linker and handled separately. During migration only those two sections are copied to the target node, and the one containing the pointers is adjusted to the new memory layout.

Frequently, dynamic memory is used, which is allocated at run-time from heap space. To support the migration of dynamically allocated memory, we built two wrapper functions (`migmem_malloc` and `migmem_free`). Two extra bytes are used to add the newly allocated memory block to a linked list, which is managed by our framework and assigned to the module. Finally, when implementing a linked list and similar complex dynamic data structures, it is very likely that pointers reside inside the dynamically allocated memory. We provide a function (`migmem_register`) to make pointers placed in heap memory known to our framework. It saves the address of the pointer in a special array. The memory for the array is also taken from heap memory and dynamically adjusted in size. The list of registered pointers is transmitted and the pointers adjusted during the migration process. Of course, it is also possible to "unregister" a pointer. Furthermore, pointers are automatically removed from the list when freeing memory containing a registered pointer.

Preparation. Although we are able to migrate variables and execution state, it is not possible to migrate state that is directly bound to the node itself, such as a network connection and a file handle. For this reason, a module is informed by a `MIG_REQUEST` event that it is about to be migrated. It then has the option to take appropriate actions, e.g., to close open sockets, before it is moved to a new node. In the case of ongoing communication with external hardware, the module is able to postpone migration by calling `mig_delay()`. It will automatically get a new migration request a few seconds later. The module can also deny migration by calling `mig_deny()`.

Migration. Before the actual migration, the module is linked, but not started on the target node. The linker already allocates both memory sections containing the data and pointers. The source node serializes all memory blocks and also transmits the old memory address to the target. The static sections are copied to the memory allocated by the linker while memory for the other blocks is allocated from the heap memory. The old addresses are used to build a lookup table to map the addresses of the source node to the target node. Using this lookup table, the pointers in the pointer section are adjusted. In a next step, the list of pointers registered at runtime is transmitted. The lookup table must be used to find their new location, before they can be adjusted. Finally, the state of the Protothread is received and the local thread structure is updated accordingly.

Continuation. After the module is successfully migrated to the new node, it continues to run and receives a `MIG_SUCCESS` event so it can reestablish its connections and perform other preparations, e.g., initialize variables omitted from migration. The thread on the old node will be terminated and its memory freed. If an error occurred during migration, e.g., as there is not sufficient memory on the target node, the migration is aborted and the module continues to run at the original node. To notify the module that the migration was aborted, it receives a `MIG_FAILED` event.

4.2 Application Example

To illustrate our programming support, we describe an application example based on the environmental monitoring stream processing example presented in Section 2.1. We concentrate on the `AGGREGATE` operator as this one is migrated in the scenario.

Fig. 3 shows the simplified listing of the `AGGREGATE` operator. It takes a number of samples (`window`), calculates the average, and forwards the result. The number of samples taken to calculate the average can be adjusted at runtime. Therefore, the memory used to save these variables is dynamically allocated. Our data stream framework abstracts the network traffic and sends commands either directly to an operator or broadcasts incoming data to all operators hosted by the node.

In the first three lines, the necessary variables are defined. The `in`-variable is a pointer and is therefore marked as such. The other two save the window size and the position to write the next incoming data. Lines 5-7 and 23 contain macros generating the structures needed by the Contiki OS to manage the Protothread. In line 11, the operator waits for an incoming event. If the event is a command, the data pointer contains additional data. If the operator is instructed to resize its window size, the old memory is freed (line 16) and a new memory is allocated (line 17). For simplicity of the example, the data is lost upon window resize. As connection handling and all further system-dependent tasks are performed by our framework, there is no need to inform the operator about a migration. Thus, due to migration support of statically as well as dynamically allocated memory, a migration is fully transparent to the operator.

```
MIGRATABLE_POINTER static u16 * in;
MIGRATABLE static u8 pos;
MIGRATABLE static u8 window;

PROCESS(p_migagg, "Mig.Aggr");
AUTOSTART_PROCESSES(&p_migagg);
PROCESS_THREAD(p_migagg, ev, data)
{
  PROCESS_BEGIN();
    while(1) {
      PROCESS_WAIT_EVENT();                          // Wait for an event
      if(ev == EV_MODULE_CMD) {                      // A command event
        if(*data == MOD_CMD_SIZE) {                  // Resize window command
          window = *(++data);                        // Set window size
          pos = 0;                                   // Reset write position
          if(in != NULL) migmem_free(in);            // Free old window
          in = migmem_alloc(window * 2);             // Allocate new window
        }
        else if(ev == EV_MODULE_DATA) {              // Handle incoming data
          in[pos++ % window] = *(u16 *)data;         // Copy data
          // Calculate average and send it
    } } }
    PROCESS_END();
}
```

Fig. 3. Simplified listing of the data stream AGGREGATE module

5 Evaluation

While our implementation runs on the native TelosB hardware, we performed most of our evaluations on top of the Cooja [19] simulator. Cooja utilizes the MSPsim simulator [20], an instruction level simulator for the MSP430 micro controller. Thus, code can be added and executed at runtime. For our evaluation, we considered the basic characteristics of our binary format, the linking process and the support for migrating stateful modules.

5.1 Support for Linking and Loading

Overall Resource Demand. Our system support for sending, receiving, linking, starting and stopping modules as well as migration is 7 KB in size and has a memory footprint of 160 B. It also includes helper functions such as for remote monitoring (e.g., listing the currently installed modules) as well as commands for managing the external flash.

Symbol Table Footprint. To measure the memory savings provided by our compression of the symbol table we analyzed the symbols of a "hello world" kernel using the default kernel for the TelosB platform, having 316 symbols in total (see Table 1).

The Contiki ELF-Linker stores all kernel symbols in an array containing the address and a pointer to the string. While this improves performance when searching for a certain symbol, we omit this additional pointer, which saves two bytes per symbol. The reason why the use of the Minilink format reduces the size not by 623 B (symbols $\times$ 2), but only 573 B, is because of our slightly larger header and 4 symbols[2] that are excluded

[2] __bss_size, __data_load_start, __data_size and __stack

Table 1. Comparison of the Contiki symbol table and the Minilink symbol table

	Size	Saved (relative)	Saved (total)
Contiki	5732 B		
Minilink	5159 B	10.00 %	10.00 %
Minilink+option(`prefix`)	3399 B	34.12 %	40.70 %
Minilink+option(`prefix,7bit`)	3083 B	9.30 %	46.21 %
Minilink+option(`prefix,7bit,offset`)	2918 B	5.35 %	49.09 %

in the Contiki symbol list. The largest savings are achieved using our simple prefix-compression (`prefix`). As the character set for symbols is limited to 7 bit ASCII, we can use the 8th bit to substitute the NULL-terminator (`7 bit`). Finally, we are able to save some extra bytes by using relative addressing where possible (`offset`). In sum, our symbol table is almost half the size of the Contiki symbol table. This speeds up the linking process as only half the data must be transferred from the external flash. These savings are even more valuable if the symbol table must be saved together with the kernel on internal flash, e.g., if there is no external flash available.

Minilink Module Footprint. We also compared the size of ELF modules provided by Contiki and their optimized variant CELF with our Minilink format (see Table 2). As a first sample, we chose a *hello_world* module, which basically outputs a "Hello World" string. The next sample is a module implementing the *AGGREGATE* operator, as outlined in the stream processing example (*mod_agg*). For evaluating a larger module, we also linked the *rudolph2* (and polite) network protocol [21] to the *AGGREGATE* operator as one module. Further on, we added a call to `watchdog_periodic()` to this module to evaluate the symbol table lookup performance (see following paragraph on Minilink linker performance for details).

The code size represents the size of the program and the data section (not `.bss`). This is the minimum size even a prelinked module must have without its header. We also list the number of used symbols and relocations. The numbers in braces represent the overhead compared to the minimum prelinked module. Apparently, the ELF has a big static overhead. This has a huge impact on small modules, as can be seen for the `hello_world` example, with an overhead of over 900 %. While CELF performs much better, Minilink has an overhead of only about 30 %.

Minilink Linker Performance. Table 3 compares the time required to link a module using the previously introduced examples. We measured the duration to build up the address index and the linking process independently. Contiki OS provides the

Table 2. Comparison of ELF modules and Minilink modules for modules of three different sizes

Application	**hello_world**	**mod_agg**	**mod_agg + rudolph2**
code size	74 B	910 B	2230 B
symbols	1	15	36
relocations	5	48	145
ELF (overhead)	752 B (+916 %)	2956 B (+225 %)	6028 B (+170 %)
CELF (overhead)	179 B (+142 %)	1611 B (+77 %)	3793 B (+70 %)
Minilink (overhead)	96 B (+30 %)	1164 B (+28 %)	2772 B (+24 %)

Coffee file system, which has a high standard deviation when opening file handles ($132 \pm 36.6\mathrm{ms}$ according to Tsiftes et al. [22]) therefore we excluded these from our measurements. However, the time to access the actual data on the external flash is included. When calculating the percentage of relocations, it has to be taken into account that each relocation results in a memory address and is therefore two bytes in size. It can be seen, that the time to build up the address index does not entirely depend on the number of different symbols, but where the last used symbol is located in the symbol table. The reason for this, are the sorted symbols. Once a symbol is found, the next symbol in the module must be further down the symbol table and there is no need to restart the search. The linking process itself scales with the size of the module. This is mainly due to the slow access to flash. Half the time is spent writing to internal flash.

5.2 Stateful Migration

The migration support is responsible for 2 of the 7 KB needed for the whole framework. It includes all functions required for the serialization of modules.

We analyzed the migration of the *mod_aggr*-module. It has 8 B of data, one pointer and 10 B allocated from heap, which totals in 20 byte that must be migrated. The size of the serialized data was 44 byte. This is mainly caused by static overhead, like the individual sizes and addresses of the different memory sections ($6 \times 4\,\mathrm{B}$), the total number of dynamically allocated memory blocks and registered pointers. The only additional overhead is 4 byte (address and size) for every additional dynamically allocated memory block.

A single-hop migration takes about 1 s. Almost all of the time is spent due to the network stack: To save power, each node has a short listening period every 300 ms. Also, the route is not previously known to the mesh network stack and has to be found first. Even under optimal conditions (route known, and the target listening), the migration would take at least 19 ms. Thereby, less then 2 ms is caused by our migration support. The serialization itself takes about 0.4 ms and the deserialization – including the allocation of additional memory and adjustment of the pointer – takes 0.6 ms.

We also compared our approach to the one suggested by [12]: Their framework provides a buffer and the module does the serialization by itself. We implement a very simple serialization using `memcopy()` and all variables placed in a struct without any error handling. This approach adds another 183 B to the *mod_aggr* Minilink module (+208 B in the case of ELF).

Table 3. Comparison of the time to build the address index and linking for different modules

Module	**hello_world**	**mod_agg**	**mod_agg + rudolph2**
Symbols	1	14	27
Last symb.	puts	rimeaddr_copy	watchdog_periodic
Match symb.	55 ms	82 ms	114 ms
Size (text + data)	78 B	910 B	2230 B
Relocations	5 (12.8 %)	48 (10.6 %)	119 (10.7 %)
Link	5 ms	67 ms	159 ms
Bytes/ms	14.4	13.6	14.0

6 Related Work

We approach related work by examining how code modules are deployed and updated in WSNs and then investigate systems supporting state migration.

Loading Code at Runtime. Using a Virtual Machine (VM) is one of the most flexible solutions for code deployment as it typically employs some form of high-level byte code that is less machine and OS dependent than plain native code. Furthermore, due to its higher expressiveness, the code size of a module is usually smaller and consequently less data needs to be transferred. In consequence, several VMs are available for sensor networks, e.g., Maté [9] and VM* [23]. In [8], VMs are compared against runtime linking of native modules, showing that installing a module using a VM can be more efficient than linking in terms of integration time, but offers less performance at runtime. Our work reduces the downsides of runtime linking while retaining the performance offered by native execution.

SOS [6] is one of the first WSN-class OSs that has native support for modules. Thus, single modules can be added without affecting the rest of the system. A limited number of kernel functions are exported using a jump table. All other function addresses are resolved at runtime using string comparison and a lookup table. Once resolved, function addresses are cached to speed up calling functions. Still, this entails a runtime overhead for every function call. Further on, all strings that are needed to resolve function calls must be saved in the internal flash.

Dunkels et al. [8] have built a runtime linker for ELF-based modules on top of the sensor OS Contiki. However, the ELF binary format is rather resource consuming. The symbol table for a standard Contiki kernel requires about 5 KB of flash memory and an ELF-module is normally more than twice the size of the resulting binary file. Accounting these facts, Dunkels et al. proposed an ELF inspired binary format called CELF, which is optimized for 16 bit microcontrollers. Dong et al. [24] also proposed an optimizing the ELF format that kept the basic design. However, they pre-fill the placeholders of a module with concrete addresses. This combines the advantages of pre- and runtime linking: If the correct kernel is installed, the module can be copied without linking; otherwise, the module can still be linked.

Based on the BTnut[3], we have built a host-based linker, which generates binary modules that are custom-tailored for a remote node [7]. This solution misses the flexibility provided by a node-based linker. However, only the final code has to be transferred, which further on can be directly executed. In sum, none of the presented approaches offers such a highly optimized binary layout for native modules like Minilink and a tailored memory-efficient linking process.

Stateful Migration. Chien-Liang et al. [13] proposed system support enabling the mobile agent paradigm in sensor networks. This was achieved using a VM concept with a dedicated instruction set that is based on Maté [9]. This way, code can be immediately executed; and it is portable so it can be shared between different nodes. However, this higher layer of abstraction adds a considerable run time overhead.

The closest related work to our knowledge is UbiMASS [12]. UbiMASS is a framework for Contiki OS supporting migration of agents at runtime – something, that

[3] http://www.btnode.ethz.ch/

can easily implemented with our approach. We believe our approach is superior in three main aspects: Firstly, the Contiki linker, which's file format we have proven inefficient, is used. Secondly, the developer of the agent must do the serialization of data. Meaning, instead of just marking each variable as migratable it must be passed to special function and recovered manually after migration. Thirdly, UbiMASS only supports the migration of integer and string variables, while we can migrate any data type and even pointers if there were made known to the framework.

7 Conclusion

In-network processing of sensor data is a powerful technique to enable long-term monitoring using WSNs. However, changing runtime-conditions demand for flexible state-preserving relocation of pre-processing modules to enable continuous monitoring. We presented a resource-efficient solution based on stateful mobile modules. At its core, we provide a size-optimized binary format and a memory-efficient linking process together with runtime support to migrate statically allocated variables as well as dynamically allocated memory. Finally, stateful mobile modules can be seen as a novel approach, which give application programmers the convenience to implement migration-enabled stateful applications using a simple API.

References

1. Hart, J.K., Martinez, K.: Environmental sensor networks: A revolution in the earth system science? Earth-Science Reviews 78, 177–191 (2006)
2. Gehrke, J., Madden, S.: Query Processing in Sensor Networks. IEEE Pervasive Computing 3, 46–55 (2004)
3. Edara, P., Limaye, A., Ramamritham, K.: Asynchronous in-network prediction: Efficient aggregation in sensor networks. ACM Transactions on Sensor Networks (TOSN) 4, 1–34 (2008)
4. Zhu, Y., Rundensteiner, E., Heineman, G.: Dynamic Plan Migration for Continuous Queries over Data Streams. In: ACM SIGMOD Conference 2004, Paris, France, pp. 431–442 (2004)
5. Jeong, J., Culler, D.: Incremental Network Programming for Wireless Sensors. In: 1st IEEE International Conference on Sensor and Ad hoc Communications and Networks (IEEE SECON 2004), Santa Clara, CA, USA (2004)
6. Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes. In: Proceedings of th 3rd ACM International Conference on Mobile Systems, Applications, and Services (ACM MobiSys 2005), Seattle, WA, USA, pp. 163–176 (2005)
7. Dressler, F., Strübe, M., Kapitza, R., Schröder-Preikschat, W.: Dynamic Software Management on BTnode Sensors. In: 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (IEEE/ACM DCOSS 2008): IEEE/ACM International Workshop on Sensor Network Engineering (IWSNE 2008), pp. 9–14 (2008)
8. Dunkels, A., Finne, N., Eriksson, J., Voigt, T.: Run-time dynamic linking for reprogramming wireless sensor networks. In: 4th ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, CO, pp. 15–28 (2006)
9. Levis, P., Culler, D.: Maté: a tiny virtual machine for sensor networks. ACM SIGOPS Operating Systems Review 36, 85–95 (2002)

10. Brouwers, N., Langendoen, K., Corke, P.: Darjeeling, a feature-rich vm for the resource poor. In: 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009), pp. 169–182. ACM, New York (2009)
11. Felser, M., Kapitza, R., Kleinöder, J., Schröder-Preikschat, W.: Dynamic Software Update of Resource-Constrained Distributed EmbeddedSystems. In: IFIP International Embedded Systems Symposium (IESS 2007), Irvine, CA, USA, vol. 231, pp. 387–400 (2007)
12. Bagci, F., Wolf, J., Ungerer, T., Bagherzadeh, N.: Mobile Agents for Wireless Sensor Networks. In: International Conference on Wireless Networks (ICWN 2009), Las Vegas, NV, USA (2009)
13. Fok, C.L., Roman, G.C., Lu, C.: Rapid development and flexible deployment of adaptive wireless sensor network applications. In: 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005), Columbus, OH, USA, pp. 653–662 (2005)
14. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: 1st IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, FL (2004)
15. Dressler, F., Kapitza, R., Daum, M., Strübe, M., Schröder-Preikschat, W., German, R., Meyer-Wegener, K.: Query Processing and System-Level Support for Runtime-Adaptive Sensor Networks. In: 16. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2009), Kassel, Germany, pp. 55–66. Springer, Heidelberg (2009)
16. TIS Committee: Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification (1995)
17. Fuggetta, A., Picco, G., Vigna, G.: Understanding code mobility. IEEE Transactions on Software Engineering 24, 342–361 (1998)
18. Dunkels, A., Schmidt, O., Voigt, T., Ali, M.: Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In: 4th ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, CO (2006)
19. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level simulation in cooja. In: European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands (2007)
20. Eriksson, J., Dunkels, A., Finne, N., Österlind, F., Voigt, T.: Mspsim – an extensible simulator for msp430-equipped sensor boards. In: European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands (2007)
21. Dunkels, A.: Rime — a lightweight layered communication stack for sensor networks. In: European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands (2007)
22. Tsiftes, N., Dunkels, A., He, Z., Voigt, T.: Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In: 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009), San Francisco, USA (2009)
23. Koshy, J., Pandey, R.: Vmstar: synthesizing scalable runtime environments for sensor networks. In: 3rd International Conference on Embedded Networked Sensor Systems (SenSys 2005), San Diego, CA, USA, pp. 243–254 (2005)
24. Dong, W., Chen, C., Lie, X., Bu, J., Liu, Y.: Dynamic Linking and Loading in Networked Embedded Systems. In: 6th IEEE International Conference on Mobile Ad Hoc and Sensor Systems 2009 (MASS 2009), Macau SAR (2009)

Design and Implementation of a Robust Sensor Data Fusion System for Unknown Signals

Younghun Kim, Thomas Schmid, and Mani B. Srivastava

Electrical Engineering Department
University of California, Los Angeles
{kimyh,thomas.schmid,mbs}@ucla.edu
http://nesl.ee.ucla.edu

Abstract. In this work, we present a robust sensor fusion system for exploratory data collection, exploiting the spatial redundancy in sensor networks. Unlike prior work, our system design criteria considers a heterogeneous correlated noise model and packet loss, but no prior knowledge of signal characteristics. The former two assumptions are both common signal degradation sources in sensor networks, while the latter allows exploratory data collection of unknown signals. Through both a numerical example and an experimental study on a large military site, we show that our proposed system reduces the noise in an unknown signal by 58.2% better than a comparable algorithm.

Keywords: Exploratory Data Collection, Robust Distributed Sensing, Data Fusion.

1 Introduction

Battery powered wireless sensing systems allow us to observe unknown signals in a large field of interest. A popular practice to collect information about the unknown signals involves (1) deploying a set of wireless sensor nodes, (2) collecting data, and (3) performing data analysis. This approach is particularly useful in two scenarios: exploratory sensing applications where no prior knowledge about the signal exists, and collecting training data for a sensing system prior to actual deployment. Designing an event detector using hypothesis testing is a good example. To design the detector, event signatures, such as statistical significance, signal energy, bandwidth, and so forth, need to be identified prior to the final system deployment [13]. Another example is the observation of large scale natural phenomena. Researchers often need unbiased signals to apply their domain expertise or to study the phenomena itself. Filter design is yet another example where one first needs to identify the bandwidth and noise characteristics of the signal[1].

In small scale experiments, the initial data gathering task is usually performed in a controlled environment where the necessary signal characteristics can easily

[1] When signals of interest are well understood, one can use models. Instead, our system is used to build models for the unknown.

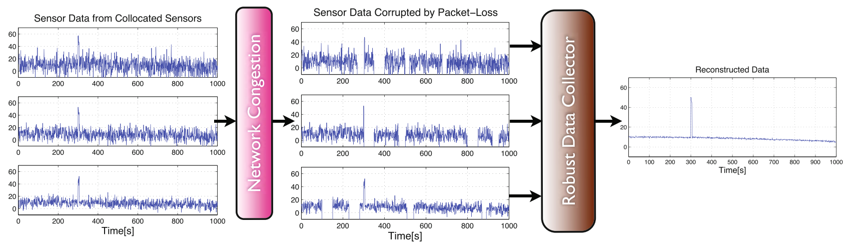


Fig. 1. Sensors suffer from different levels of noise (left). Data loss due to network congestion decreases signal integrity further (middle). The objective of our system design is to minimize the measurement noise and to cope with packet losses without knowing the characteristics of the signal (right).

be identified (we call this the sensor calibration, system training, and/or system identification phase). Unfortunately, this becomes very difficult at larger scales. Moreover, it is well understood that networked sensing systems suffer significantly from two major sources of signal degradation: packet loss due to congestion and unreliable communication links, and measurement noise due to low quality sensors.

Common solutions to this initial data gathering use well-defined signal and noise models in order to apply an effective filter. However, this modeling often introduces signal bias, while successful signal characterization needs unbiased estimation. Therefore, popular signal and noise models are to be avoided as not to add bias to the sensing system. For example, unless we pick the right bandwidth, an assumption of a *slow varying* signal compromises the integrity of the system as it cuts off high frequency components. Similarly, the popular Gauss-Markov assumption is not desired as collocated sensors can be subject to similar noise sources, resulting in correlated noise characteristics. Therefore, it is often desirable to have an initial data gathering system an unbiased distributed sensing system such that we can later on use more sophisticated algorithms [3,7,20,21,22].

This paper describes a sensing architecture for exploratory data collection and develops a robust sensor information fusion method (Figure 1). Our method exploits the added redundancy in a large scale sensor network. Our algorithm fuses the information of collocated sensors monitoring the same signal of interest by solving an optimization problem. The result is an unbiased, robust signal. Unlike other prior work, our proposed architecture tolerates heterogeneous correlated noise and missing data. We evaluate our system numerically, as well as with a real data set collected during a large military field experiment.

2 Exploratory Sensing System Design

Figure 2 depicts the sensing system architecture consisting of two tiers of nodes and a database server. The sensor nodes consist of several sensing clusters. The nodes in a cluster observe the same signal of interest, are time synchronized, and

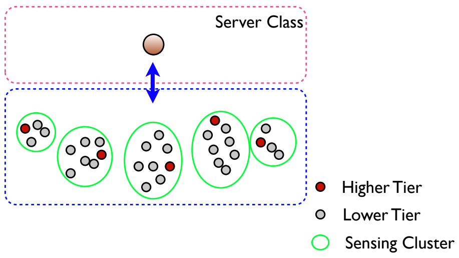


Fig. 2. Tiered Sensing Architecture. Fusing the data from each sensing cluster provides a more robust and reliable signal.

send their samples to a higher tier node also part of the cluster. The higher tier node is responsible for fusing the signals from the sensing cluster. The collocation of the sensor nodes within a sensing cluster adds *redundancy* and *robustness* towards *measurement noise* and *data loss*. The sensing cluster and the higher tier node is thus the basic sensing entity[2]. The next paragraphs describe the sensing and fusion model for this basic sensing unit; although the model looks like a centralized fusion mechanism, it applies to each small sensing cluster, and not over the whole network.

Let us define $y_i(t)$ as the measurement from a node in a sensing cluster.

$$y_i(t) = \kappa_i(t)\left(\alpha_i s(t) + \beta_i + n_i(t)\right), \tag{1}$$

where y_i is the observation of the i-th sensor, κ_i is a boolean random variable indicating missing data with probability $P(\kappa_i = 0) = p_i$, α_i is a scaling factor, β_i is an offset error, n_i is zero-mean additive noise (not necessarily Gaussian), and t is a discrete time variable. This sensing model is a generalization of the sensing model $y_i(t) = s(t) + n_i(t)$ used in [7, 20, 21, 22] adding scale α_i and offset β_i representing the effect of uncalibrated sensors, and $\kappa_i(t)$ to capture network data loss. It thus allows to have a more complex noise model, unlike the homogeneous noise model assumed in earlier work [3, 7, 15, 16].

Balzano et al. [1] showed that the scale α_i and the offset β_i can be estimated with small manual intervention. To simplify notation, but without loss of generality, we can set the observation as [3]

$$y_i = \kappa_i(s + n_i). \tag{2}$$

For succinct notation, we omit the time constant. Note that the implication of the normalization for α_i is that the noise characteristics change, i.e. the noise covariance matrix can be an arbitrary positive definite matrix. The goal of our

[2] Several sensing clusters can share the same higher tier node. But the higher tier node fuses each cluster individually.

[3] This changes the noise characteristics, which we will take into account by using its covariance matrix. Errors in this process, however, would add bias to the system, which is found to be minimal in our experiment. Nevertheless, further investigation at coping with the bias issue is of future interest.

fusion algorithm is to estimate $\hat{s}(t) \sim s(t)$ using the set of observations $Y_{obs} = \{y_i(t) : i \in \mathbb{I}\}$, where $\mathbb{I}$ is the node index set.

2.1 Fusion Mechanism Design

We consider the simple problem of a robust sensor fusion mechanism where we want to estimate the true signal s. Our problem is to find a function $f(Y_{obs})$ that minimizes the error variance and mean value of s,

$$\begin{array}{ll} \min\limits_{f} & Var(s - \hat{s}|Y_{obs}) \\ \text{s.t.} & E(\hat{s}|Y_{obs}) = s \\ \text{where} & f(Y_{obs}) = \hat{s} : \text{Estimation of } s. \end{array} \tag{3}$$

To motivate this particular problem, we want to note that the second order moments fully characterize Gaussian processes and are often sufficient to characterize stochastic processes that are not heavy tailed (e.g. the second moments exist) [12]. For example, even if the error distribution is not Gaussian, the mean and variance of a signal represent the quality of estimation (also known as the empirical risk minimization [4]).

Since problem (3) is an implicit problem, we cannot solve it without casting it into an explicit form. We follow steps to perform this transformation [4].

We choose to use the best linear unbiased filter form f for a set of observations $Y_{obs} = \{y_i : i \in \mathbb{I}\}$ because it achieves the minimum error variance [10],

$$f(Y_{obs}) = \hat{s} = \sum_{i=1}^{N} w_i y_i. \tag{4}$$

We now follow three steps to develop a new mechanism that selects the *optimal weights* that tolerates both *missing packets* and *correlated noise.*

I. Optimal Weighting. First, let us assume that we do not have missing data, i.e. $P(\kappa_i = 0) = 0$, which we will generalize later on. Thus, the observation becomes

$$y_i = s + n_i. \tag{5}$$

We then want to estimate the signal, s, by combining all the y_is. A simple, yet efficient method to estimate s is to average all the observations, i.e., $\frac{1}{N}\sum_{i=1}^{N} y_i$, regardless of the noise characteristics. In fact, many consensus algorithms have explored efficient methods for this particular problem [3, 6, 22]. However, we cannot use averaging because collocated sensors have heterogeneous correlated noise, i.e. it's covariance matrix is not necessarily a diagonal matrix.

Therefore, we formulate a different problem where we choose the optimal weights w_i given a correlated heterogeneous covariance matrix. To determine

[4] This follows Best Linear Unbiased Estimators (BLUE) [10] except that the steps explicitly incorporate missing data and non-homogeneous noise cases.

each sensor's weight, we first set up a metric that captures the quality of the estimation. The signal quality can be described by using different metrics, e.g. signal-to-noise ratio, error variance, asymptotic stability, etc. We chose to use a well established metric, the error variance[5].

Let ϵ be the estimation error defined as $\epsilon \overset{def}{=} s - \hat{s}$. It follows that

$$\begin{aligned} \epsilon &= s - \hat{s} \\ &= s - \textstyle\sum_{i=1}^{N} w_i y_i \\ &= s - \textstyle\sum_{i=1}^{N} w_i s - \sum_{i=1}^{N} w_i n_i \\ &= - \textstyle\sum_{i=1}^{N} w_i n_i \end{aligned} \tag{6}$$

Now, the second order moments of the error becomes

$$Var(s - \hat{s}|Y_{obs}) = Var(-\sum_{i=1}^{N} n_i) = \mathbf{w}^T \mathbf{\Sigma} \mathbf{w}, \tag{7}$$

where $\mathbf{\Sigma}$ is the covariance matrix of $\mathbf{n} = [n_1 n_2 ... n_N]^T$, and $\mathbf{w} = [w_1 w_2 ... w_N]^T$.

Now, the following problem chooses the optimal $\mathbf{w}$ such that the error variance is minimized.

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}^T \mathbf{\Sigma} \mathbf{w} \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{N} w_i = 1. \end{aligned} \tag{8}$$

Note that the constraint $\sum_{i=1}^{N} w_i = 1$ ensures $E(\hat{s}|Y_{obs}) = s$.

II. Coping with Missing Data. Now we shall generalize (8) to capture missing data. Missing or lost data is a common artifact in wireless sensor networks. Causes range from RF interference to sensor malfunction. Additionally, retransmission might not be feasible because it introduces an unpredictable packet delay and increases the energy consumption.

However, while data loss is unpredictable, we can detect it at the fusion center with simple measures such as message timestamps or sequence numbers. In other words, in $y_i = \kappa_i(s + n_i)$, κ_i is a random Boolean variable indicating packet loss with a probability $P(\kappa_i = 0) = p_i$. Nevertheless, we know that κ_i can be determined by the network layer.

The implication of this deterministic behavior of missing data is that the fusion node can dynamically tune the fusion parameters according to packet loss dynamics. For example, if the k-th sensor fails to transmit its packet, the solution can be formulated as the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}_{\mathbb{I}\setminus\{\mathbf{k}\}}} \quad & \mathbf{w}_{\mathbb{I}\setminus\{k\}}^T \mathbf{\Sigma}_{\mathbb{I}\setminus\{k\}} \mathbf{w}_{\mathbb{I}\setminus\{k\}} \\ \text{s.t.} \quad & \textstyle\sum_{i \in \mathbb{I}\setminus\{k\}} w_i = 1, \end{aligned} \tag{9}$$

where $\mathbb{I}$ is the sensor index set, $\mathbf{w}_{\mathbb{I}\setminus\{k\}}$ is a vector of w_i excluding w_k, $\mathbf{\Sigma}_{I\setminus\{k\}}$ is a covariance matrix of n_is except for n_k (i.e. eliminate the k-th column and row

[5] Another interpretation of the error variance is the empirical risk.

of Σ). The solution to Equation (9) is the optimal weight in case the k-th data sample is missing.

We define the following terminology to generalize this idea: Let $\Gamma \subset \mathbb{I}$ be a set of indices of missing data, $\mathbf{w}_{\mathbb{I}\backslash\Gamma}$ be a vector of w_i, where $i \in \mathbb{I}\backslash\Gamma$, and $\mathbf{\Sigma}_{I\backslash\Gamma}$ be the covariance matrix of a vector of n_i's, where $i \in \mathbb{I}\backslash\Gamma$. Using this terminology, we can rewrite the optimization problem as

$$\begin{aligned} \min_{\mathbf{w}_{\mathbb{I}\backslash\boldsymbol{\Gamma}}} \quad & \mathbf{w}_{\mathbb{I}\backslash\Gamma}^T \mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma} \mathbf{w}_{\mathbb{I}\backslash\Gamma} \\ \text{s.t.} \quad & \textstyle\sum_{i\in\mathbb{I}\backslash\Gamma} w_i = 1 \end{aligned} \tag{10}$$

The weight calculated from Equation (10) is the optimal weight vector for the measurements that are available at a given instance.

III. Robust Estimation via Quadratic Program. Let us consider the problem of a time sequence of missing data. Let $\Gamma(t) \subset \mathbb{I}$ be the set of indices of missing data, where t is the discrete time index. The Quadratic Program Equation (11) is a time-varying sequence as the sequence of the set of missing data $\Gamma(t)$[6] changes over time. Therefore, we have

$$\begin{aligned} \min_{\mathbf{w}_{\mathbb{I}\backslash\boldsymbol{\Gamma}(\mathbf{t})}} \quad & \mathbf{w}_{\mathbb{I}\backslash\Gamma(t)}^T \mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma(t)} \mathbf{w}_{\mathbb{I}\backslash\Gamma(t)} \\ \text{s.t.} \quad & \textstyle\sum_{i\in\mathbb{I}\backslash\Gamma(t)} w_i = 1. \end{aligned} \tag{11}$$

Analytic Solution Sequence We can solve (11) analytically and show that the solution to this time-varying quadratic program is the optimal spatial filter in the mean square sense.

Equation (11) is an *equality constraint convex quadratic program*. We want to find a solution that satisfies the *Karush-Kuhn-Tucker* (KKT) conditions for optimality [4]. The KKT conditions for Equation (11) are

$$\begin{aligned} &\mathbf{1}^T \mathbf{w}^*_{\mathbb{I}\backslash\boldsymbol{\Gamma}(\mathbf{t})} = 1 \\ &\mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma(t)} \mathbf{w}^*_{\mathbb{I}\backslash\boldsymbol{\Gamma}(\mathbf{t})} + \mathbf{1}\nu^* = \mathbf{0}, \end{aligned} \tag{12}$$

where ν^* is a Lagrange multiplier. This is equivalent to

$$\begin{bmatrix} \mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma(t)} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w}^*_{\mathbb{I}\backslash\boldsymbol{\Gamma}(\mathbf{t})} \\ \nu^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}.[7] \tag{13}$$

By solving this set of $(|\mathbb{I}\backslash\Gamma(t)|+1)$ linear equations the fusion mechanism obtains the optimal weights. The left matrix has a non-trivial determinant, since

$$\begin{bmatrix} \mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma(t)} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma}_{\mathbb{I}\backslash\Gamma(t)} & \mathbf{0} \\ \mathbf{1}^T & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{\Sigma}^{-1}_{\mathbb{I}\backslash\Gamma(t)}\mathbf{1} \\ \mathbf{0}^T & -\mathbf{1}^T\mathbf{\Sigma}^{-1}_{\mathbb{I}\backslash\Gamma(t)}\mathbf{1} \end{bmatrix} \tag{14}$$

[6] $\Gamma(t)$ can be seen as a realization of a stochastic process of intermittent communication failure.

[7] **1** and **0** are column vectors of 1 and 0 in an appropriate dimension, whereas 1 and 0 are scalar.

and $det(\mathbf{\Sigma}_{\mathbb{I}\setminus\Gamma(t)})det(-\mathbf{1}^T\mathbf{\Sigma}^{-1}_{\mathbb{I}\setminus\Gamma(t)}\mathbf{1}) \neq 0$. It is therefore always invertible as the covariance matrix is positive definite, and thus the optimal weight vector exists.

We can interpret this analytic solution as following. By observing the network dynamics, the fusion node *dynamically tunes the weights* for each sensor in the cluster to achieve the minimum error variance. The implementation is simple because only a matrix inversion and a multiplication are needed, whose computational complexity is $O(n^3)$, where n is the number of nodes if we use the Gauss-Jordan elimination, or it can be $O(n^{2.376})$ if we use the Coppersmith-Winograd algorithm [5].

3 Evaluation

We show two evaluation studies using a simulation result and a large-field experimental data set. In the simulation, we pick a simple signal that has both low frequency data and an impulse train emulating ambient signal and a sequence of extreme events. We show that our fusion mechanism removes noise to a certain level without compromising any signal characteristics.

In the field experiment on a large military site, we evaluated two different sensor types: humidity and magnetic field sensors. The goal of this data collection is to identify the characteristics of the signals of both natural phenomena and an event. The humidity sensor is an example sensing modality of natural phenomena. The magnetic sensor is used to identify the event signatures from a moving military tank. This experiment shows how our algorithm maintains the signal of interest, while maintaining resilience to measurement noise and packet loss.

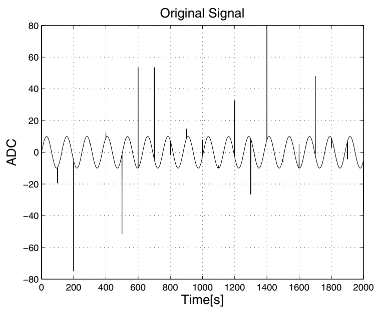
3.1 Synthetic Signal Evaluation

Figure 3a represents our synthetic sample signal, which is the sum of a low-frequency sinusoidal wave and a pulse train. The pulse train represents a sporadic event of interest and the sinusoidal wave represents ambient signal changes.

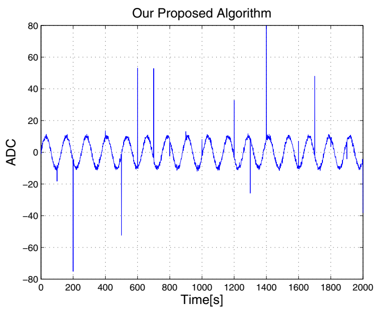
We use nine sensors with a randomly chosen covariance noise matrix to evaluate the system. Packet loss rate for each sensor node is set to 5%, and we assume that packet loss is independent [8]. Thus the probability of not getting any packets is 0.05^9, which means that the chance of not getting data from all the sensors in the numerical evaluation is extremely small.

Figure 3c represents the sampled data from the best sensor whose error variance is minimum among other sensors, i.e. the sensor with the smallest noise level within the sensing cluster, corresponding to the base line. We compare the performance of our fuser with three different filtering schemes: a conventional consensus algorithm, a steady-state Kalman filter, and a low-pass filter. Figure 3e shows the conventional consensus algorithm over the nine sensors [3]. We

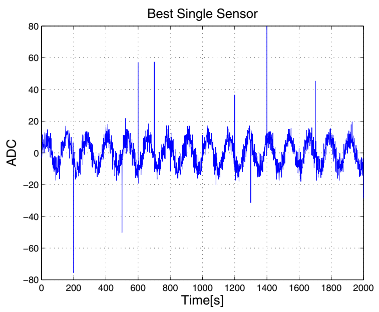
[8] Radio congestion can be more complicated [24]. While the simulation uses the simple model, our experimental data set shows our proposed architecture works in a real environment. We evaluate the implication of this later in this section.



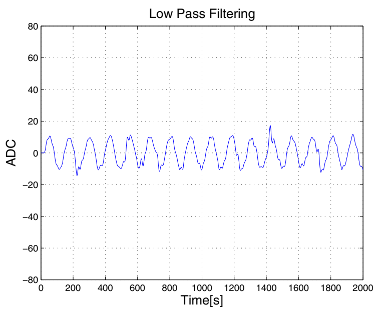
(a) Original Signal



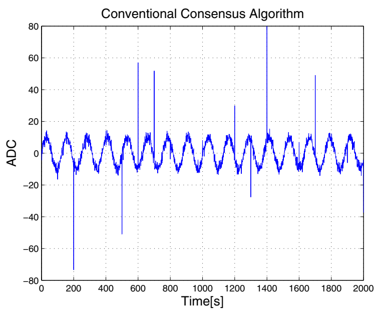
(b) Our proposed fusion algorithm



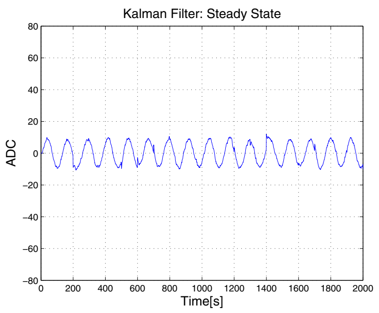
(c) Single Best Sensor



(d) Low Pass Filtering. This filtering removes the events.



(e) Conventional Consensus Algorithm



(f) Steady State Kalman Filter. The Kalman Filter removes the event signatures.

Fig. 3. Synthetic Signal evaluation. While some filters can easily reproduce the slow-varying signals, they remove the spikes.

can see that the conventional consensus algorithm significantly suppresses the noise if we compare the output with the original signal shown in Figure 3c. The problem is that some of the sensors have much more noise than the others, and the bad quality sensors and the correlated noise prevent the consensus algorithm from suppressing the noise further. In comparison, our proposed system

Table 1. Performance Comparison

Methods	Error STD	Improvement(%)
Best Sensor	4.07	0(Base)
Our System	0.79	80
Normal Consensus	1.89	53
Kalman Filter	6.09	-50
Low Pass Filter	7.07	-74

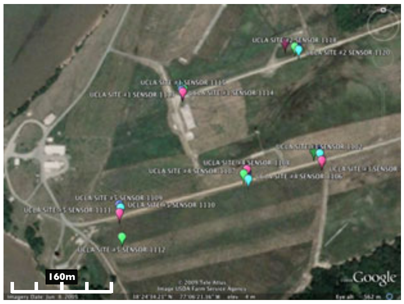
rejects noise more efficiently (Figure 3b). This is obtained because the weight associated with each sensor is optimized for the correlated noise characteristics, thus suppressing the measurement noise more efficiently from noisier sensors. In addition, notice that the event at t=1850s is much clearer in our proposed algorithm (Figure 3b) than in the conventional consensus algorithm (Figure 3e). This is because the conventional consensus algorithm is not made to deal with correlated heterogeneous noise while ours is specifically designed for it.

Note that the traditional approaches work well for the slow varying signals including a good noise level reduction. However, they also filter out most of the events, thus rendering the signals unusable for training of an event detection algorithm. For example, Figure 3d and 3f show the signals from a low pass filter and a discrete Kalman Filter realization. The low pass filtering is able to remove most of the measurement noise, though it fails to recover the spikes, which is to be expected. The optimal Kalman filter reconstructs the low frequency signal even better, but it too removes the events. Unless we know the bandwidth of the signal ahead of filtering, we can not improve the situation. Thus, our approach can reduce measurement noise and give a good idea of the signal bandwidth, such that other filters can be run on the data in a second pass.

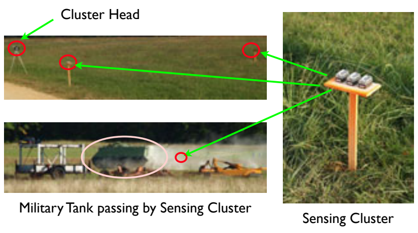
Table 1 summarizes the performance for each algorithm. (-) improvement indicates that the performance has become worse. This is partly due to the low-pass filter changing the phase of the signal. The conventional consensus algorithm has a performance increase of 53% from the base, while our approach has a performance increase of 80%. The performance gain of our proposed algorithm from the normal consensus algorithm is 58.2% (Error STD from 1.89 to 0.79). This is because the sensors with increased noise contribute equally to the fusion mechanism in the normal consensus mechanism, whereas our mechanism accounts for the sensing quality when computing the weights.

3.2 Field Experiment

Experimental Setup. Figure 4a depicts the field sensor deployment in a $108,000m^2$ military field, Southern Maryland, USA. The system consists of five sensing groups. Each group includes a cluster head and nine mote class nodes. The sensing modalities included magnetic, temperature, light intensity, acoustics, and humidity. For each sensing modality, we used three collocated nodes that synchronously sample the sensors. The cluster head runs a custom embedded Linux software on a Gumstix Verdex^{TM}. It has a long-range Wi-Fi



(a)



(b)

Fig. 4. Field Sensor Deployment Floor Plan: Each colored bubble indicates a sensing cluster (a). Collocated sensor nodes sample the same phenomena. In this example, the goal was to identify signals from a military tank driving by the sensors (b).

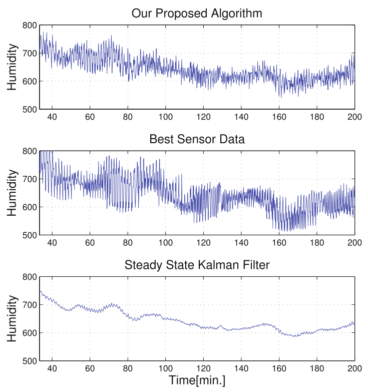
connection to a local server. Two base station nodes, which collect data from remote sensor nodes, are connected to each cluster head. Each cluster head was responsible to collect the data from three individual clusters. Each cluster consists of three sensor nodes that synchronously sample the same physical phenomena.

We used an extension of the TPSN Synchronization Protocol [9] for inter-node time synchronization, achieving < 2ms time-synchronization accuracy. The cluster head forwarded the collected data to a central server via a long range WiFi connection. The central server was equipped with a high-gain sectorized antenna to achieve the long distance connections. The physical distance between the local server and each cluster head varied between 190m to 410m. The distance between the sensor nodes and their respective cluster head varied between 2m to 50m. The central server stored the data in a custom database front-end.

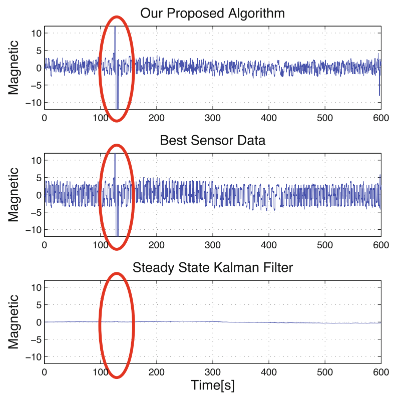
By having three identical collocated sensor nodes that synchronously sample the physical phenomena, we can employ our fusion mechanism to achieve a better signal-to-noise ratio and resilience to packet losses (Figure 4b).

Humidity Sensor Data. Because of windy conditions and the direct sun exposure of the sensors, the ambient sensors exhibit an interesting noise characteristic. The sampling interval for the humidity sensor was 30 seconds, and three identical sensors at each sensing spot (Fig. 4a) sampled their sensors at the same time. Figure 5a shows the noisy behavior from a humidity sensor. By applying the proposed fuser, we can reduce the noise (top graph in Figure 5a).

However, a simple Kalman Filter realization can provide a better quality of information as such natural phenomena are slow-varying, and the basic knowledge of the weather patterns is useful for their design. As we can see in Figure 5a, the Kalman filtered signal gives a good understanding of the field data. However, the Kalman Filtering can be tuned even better knowing the specific characteristics of the weather conditions in the field. For example, if we fine-tune the Kalman Filter by using the data from our robust fuser, an even better signal-to-noise ratio (23% improvement) can be achieved. This shows the utility of our robust fuser as a first filter of exploratory data gathering.



(a) Field Humidity Data Trace



(b) Field Magnetic Data Trace for tank detection

Fig. 5. For slow-varying signals, the Kalman filter works well even if the modeling is not precise (a). This shows that our proposed algorithm suppresses measurement noise, while preserving the event signature from the tank (b). The Kalman filter effectively removes measurement noise but it also filters out the event signature (bottom of (b)).

Magnetic Sensor Data. For identifying unknown event signatures, our system is much more valuable as we do not have prior knowledge of the signature. In this data, the sampling rate for the magnetic sensors was 1Hz, which is at the limit for detecting a military tank. Three identical magnetometers are synchronized and sample the magnetic field at the same time. However, unlike ambient temperature or humidity, the movement of a vehicle can be sudden, and therefore we don't want to remove this event with improper filtering. Figure 5b illustrates the case of a tank passing our sensors cluster. Our proposed algorithm minimizes the measurement noise while maintaining the event signature. As we can see from the bottom plot of Figure 5b, it is difficult to design a good Kalman filter as the dynamics of the signal is not known. As a result, it filters out the signature of the vehicle passing event, which means the inadequately designed filter could fail to meet the application requirement.

3.3 Empirical Study on a Redundancy and Performance Trade-Off

A valid practical question is how many sensors are necessary in order to optimize the outcome of the fuser performance? If the communication channel had infinite capacity, adding an infinite number of sensors would become better and better.

In reality, however, adding more sensor nodes congests the wireless channel. Therefore, there is a break-even point where more sensors marginally increase the performance. Although this is not the main focus of this paper, we conduct an additional experiment on the platform deployed in the field experiment to investigate the trade-offs.

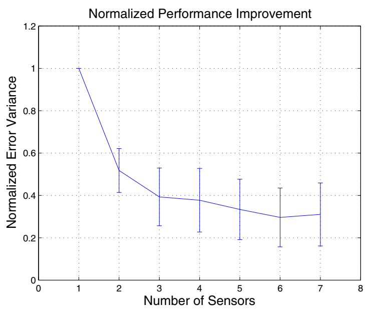


Fig. 6. Trade-off between sensor redundancy and channel congestion: With more than three sensors, packet loss due to congestion degrades any possible gain

Each sensor node was sampling light intensity at 35Hz and sent the samples over the radio to the cluster head. For each experiment we change the number of nodes and compute the normalized noise variance after the fusion mechanism is applied to the data. Figure 6 illustrates the performance improvements with an increasing number of sensors. The noise level decreases with the increasing number of nodes. But with more than three sensors, packet loss due to congestion degrades any possible gain. Note that the optimal number of nodes changes depending on the network traffic. We leave a further investigation of this problem to future work.

4 Related Work

While our paper tackles a specific sensing problem, it is part of a larger class of information fusion problems in sensor networks. Information fusion in sensor networks has a broad context varying from event detection to high-level contextual information inference. For a good summary, Nakamura et al. [14] provide a comprehensive collection of information fusion mechanisms in sensor networks.

In the context of robust signal fusion, control theory is concerned highly with reliable estimation and detection. The theory usually exploits the laws of probability to compute a state vector from a sequence of measurement vectors that are corrupted by measurement noise. In control theory, many state estimators [8] make use of well-defined system dynamic models. By using the systems dynamic model, one can design a state observer. Recently, Sinopoli et al. [19] developed an extension to the Kalman Filter that estimates states with intermittent observation loss. They show the conditions under which the estimation error covariance is bounded. It uses a state-space representation to model a dynamic system, and designs a robust estimation method. Our system is different in that we do not use any knowledge about the dynamics of the system.

The average consensus algorithm [3] has been widely used for aggregation in sensor networks. Several publications [7, 20, 21, 22] developed different versions of the average consensus algorithm. The basic idea is to average all the sensor readings from multiple sensor nodes to achieve a better quality of signal from noisy measurements. Many papers have addressed the development of efficient distributed algorithms. For example, the Gossip algorithm [3] has received attention in sensor network applications, because of its simplicity and robustness in noisy and uncertain environments. In a similar context, Speranzon et al. [20] have proposed an adaptive consensus algorithm. Dimakis et al. [6] have developed an efficient method that minimizes the number of packet exchanges for the distributed consensus mechanism by exploiting a geographical topology among sensor nodes. More recently, Xiao et al. [22,23] have developed a distributed consensus algorithm that estimates the global state information under a network topology. All these methods deal more about in-network signal processing while our system is used to understand the unknown signals first.

Krause et al. [11] developed a spatial interpolation method that minimizes the number of sensors to cover a field of interest. They use the fact that the temperature field change in a building is a Gaussian process where values between sensor nodes can be approximated. Their work intended to minimize the number of sensors and thus redundancy, while our approach explicitly makes use of redundancy for robust data fusion.

Rao et al. [16,17] discuss a fusion algorithm that combines information from various sensors to reliably detect events. They show that a simple fusion algorithm performs better than a single expensive sensor. Our approach is different because their algorithm uses a set of wired sensors and an i.i.d. training sample hence does not incorporate the loss of data packets nor identify unknown signals.

5 Conclusion

We proposed a distributed sensing architecture and fusion mechanism that fuses synchronous samples from collocated sensor nodes. By exploiting spatial correlation and redundancy, our system minimizes the second order moments of measurement noise in the presence of packet losses. Unlike prior work, our fusion design does not assume any temporal signal characteristics, an important property for exploratory data collection and sensing system training, where the observed signals are unknown. Additionally, we use a heterogeneous noise model to account for different and cross-correlated noise characteristics across distributed sensors in the tiered sensing architecture. The solution is simple (Equation (13)) consisting of a matrix inversion and multiplication. Both simulation and experimental data show that our system achieves these properties well under significant noise and unreliable communication environments.

Acknowledgments

The authors would like to thank Zainul M. Charbiwala and Rahul Balani for their help in the field experiment. This research was sponsored by US Army Research laboratory and the UK Ministry of Defense and was Normalized Error

Variance accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The U.S. and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Balzano, L., Nowak, R.: Blind calibration of sensor networks. In: IPSN (2007)
2. Bellman, R.: Dynamic Programming. Dover Publications, Mineola (2003)
3. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Gossip algorithms: Design, analysis and applications. In: IEEE INFOCOM (2005)
4. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge Press, Cambridge (2004)
5. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9 (1990)
6. Dimakis, A., Sarwate, A., Wainwright, M.: Geographic gossip: efficient aggregation for sensor networks. In: IPSN (2006)
7. Fischione, C., Speranzon, A., Johansson, K.H., Sangiovanni-Vincentelli, A.: Peer-to-peer estimation over wireless sensor networks via lipschitz optimization. In: IPSN (2009)
8. Franklin, G.F., Powell, J.D., Emami-Naeini, A.: Feedback Control of Dynamic Systems. Prentice-Hall, Englewood Cliffs (2002)
9. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync protocol for sensor networks. In: SenSys (2003)
10. Kay, S.M.: Fundamentals of Statistical Signal Processing. Prentice-Hall, Englewood Cliffs (1993)
11. Krause, A., Guestrin, C., Gupta, A., Kleinberg, J.: Near-optimal sensor placements: maximizing information while minimizing communication cost. In: IPSN (2006)
12. Mallat, S., Papanicolaou, G., Zhang, Z., Mallat, S.E.: Adaptive covariance estimation of locally stationary processes. Ann. Statist. 26, 1–47 (1995)
13. McDonough, R.N., Whalen, A.: Detection of Signals in Noise, 2nd edn. Academic Press, London (1995)
14. Nakamura, E.F., Loureiro, A.A.F., Frery, A.C.: Information fusion for wireless sensor networks: Methods, models and classifications. ACM Comput. Surv. 39(3), 9 (2007)
15. Olfati-Saber, R., Shamma, J.: Consensus filters for sensor networks and distributed sensor fusion. In: IEEE CDC-ECC (2005)
16. Rao, N.: On fusers that perform better than best sensor. In: IEEE Trans. on PAMI (2001)
17. Rao, N., Oblow, E., Glover, C., Liepins, G.: N-learners problem: fusion of concepts. In: IEEE Trans. on Systems, Man and Cybernetics (1994)
18. Scharf, L.: Statistical Signal Processing. Prentice-Hall, Englewood Cliffs (1990)

19. Sinopoli, B., Schenato, L., Franceschetti, M., Poolla, K., Jordan, M., Sastry, S.: Kalman filtering with intermittent observations. IEEE Trans. on Automatic Control (2004)
20. Speranzon, A., Fischione, C., Johansson, B., Johansson, K.: Adaptive distributed estimation over wireless sensor networks with packet losses. In: IEEE CDC (2007)
21. Speranzon, A., Fischione, C., Johansson, K., Sangiovanni-Vincentelli, A.: A distributed minimum variance estimator for sensor networks. IEEE Journal on Selected Areas in Communications (2008)
22. Xiao, L., Boyd, S., Kim, S.-J.: Distributed average consensus with least-mean-square deviation. J. Parallel Distrib. Comput. 67(1), 33–46 (2007)
23. Xiao, L., Boyd, S., Lall, S.: A scheme for robust distributed sensor fusion based on average consensus. In: IPSN (2005)
24. Zhao, J., Govindan, R.: Understanding packet delivery performance in dense wireless sensor networks. In: SenSys (2003)

Control Theoretic Sensor Deployment Approach for Data Fusion Based Detection

Ahmad Ababnah and Balasubramaniam Natarajan[⋆]

Kansas State University, Manhattan KS 66506, USA
ababnah@ksu.edu, bala@ksu.edu

Abstract. In this paper, we study the sensor deployment problem in a value fusion based distributed sensor network (DSN) detection system. More specifically, we study the problem of determining the positions at which a fixed number of sensors can be deployed in order to minimize the squared error (SE) between achieved and required detection probabilities while satisfying false alarm requirements. We show that this deployment problem can be modeled as a linear quadratic regulator problem (LQR). Subsequently, we develop two deployment algorithms; an optimal control based and a suboptimal deployment algorithm. We compare the performance of the proposed algorithms to that of a greedy deployment algorithm. Results indicate that the proposed algorithms have a faster SE convergence rate than that of the greedy algorithm. As a result, the proposed algorithms can use as much as 25% fewer number of sensors than the greedy algorithm to satisfy the same detection and false alarm requirements.

1 Introduction

Distributed sensor networks (DSN's) have a wide range of applications. One of these applications is in detection and surveillance systems. In these systems, the presence of a target/phenomena is detected by processing information (e.g., measurements) provided by sensors that are deployed within the area of interest. As in any detection system, the system's performance can be characterized in terms of false alarm and detection probabilities. In practice, a detection system is designed such that upper bounds on false alarm probabilities and lower bounds on detection probabilities are met. In systems where the target information obtained by a sensor depends on its relative position to the target, the spatial distribution of sensors plays an important role in determining the detection performance of the system.

The problem of minimizing the number of sensors required to satisfy the detection requirements was examined in [1] and [2]. The detection rule was such that, if at least one sensor reports a detection decision then a target was assumed detected. This is a simple detection rule, in which effectively sensors do

[⋆] This work was partially funded through the award of a contract from the Marine Corps Systems Command (MCSC) to M2 Technologies Inc.

not collaborate. Moreover, in [1] and [2], false alarm requirements were not considered. For a distributed detection system (i.e., multiple sensors), data fusion methods [3] are used to simultaneously meet both false alarm and detection probability requirements. In [4], the sensor deployment problem was examined with the goal of minimizing the number of sensors needed to meet both false alarm and detection requirements. Sensors measure energy emitted within their vicinity and report the noisy energy measurements to a fusion center (FC). The FC uses value fusion, which is a simple data fusion method, in order to decide on the presence/ absence of the target or phenomena. The resulting false alarm and detection probabilities are nonlinearly related to the number and positions of the deployed sensors, which complicates the treatment of the problem. The authors propose a stochastic optimization algorithm called the D&C algorithm. This algorithm is heuristic in nature and can not guarantee optimality of resulting solutions. Additionally, it does not accommodate systems with non-uniform false alarm/detection requirements.

In this paper, our goal is to deploy a fixed number of sensors such that the squared error (SE) between achieved and required detection probabilities within the area of interest is minimized, while false alarm requirements are met. We consider a collaborative DSN system, in which sensors take amplitude measurements that are corrupted by additive Gaussian noise and report these measurements to an FC, where a value fusion detection rule is employed. We propose a novel sequential sensor deployment framework that employs concepts from optimal control theory. In particular, we model the deployment problem as a linear quadratic regulator (LQR) problem. This is achieved by approximating the evolution of the difference between achieved and required detection probabilities as a linear function of the position of the sensor being deployed. In our LQR formulation, the control vectors at each discrete step in the evolution of the system correspond to the positions of the sensors being deployed. Additionally, the SE between achieved and required detection probabilities corresponds to our LQR cost function, which we need to minimize. However, we do not incur penalty for satisfying/exceeding the detection requirements. False alarm probability requirements are not incorporated in the cost function, since they can be satisfied by choosing a suitable detection threshold at the FC after each sensor deployment. In an LQR problem, the optimal control vectors minimize the cost function and can be calculated by solving a set of optimality conditions (i.e., Karush-Kuhn-Tucker (KKT) conditions). These conditions can be solved using the sweep method, which is commonly used to solve for the optimal control vectors in an LQR problem.

2 System Model

The area of interest is modeled as a grid $\mathcal{G}$ of $N_x \times N_y$ points. Although the target can be any where in the area of interest, we focus our attention at detecting the target at grid points. By increasing the number of grid points, the resolution of target detection can be improved. Each point is associated with a pair of false alarm and detection probability requirements. False alarm and detection requirements can be arranged in two $N_xN_y \times 1$ vectors $\mathbf{p}_f^{req}$ and $\mathbf{p}_d^{req}$, respectively. Sensors are allowed to be deployed on the grid points in order to detect the presence (hypothesis H_1) or absence (hypothesis H_0) of a target/phenomena. We assume that sensors measure the amplitude of the signal emitted by the target/phenomena and that each measurement is corrupted by additive Gaussian noise. Therefore, the measurement (U_i) of the i-th sensor, under the two hypothesis, is given as

$$U_i = N_i \mid H_0 \text{ true} \tag{1}$$

$$U_i = A(d_i) + N_i \mid H_1 \text{ true} \tag{2}$$

where, $N_i \sim \mathcal{N}(0, \sigma^2)$. Furthermore, we assume that sensor measurement noise is i.i.d. The signal amplitude $A(d_i)$ is distance dependent, and is given as

$$A(d_i) = \begin{cases} A_0 & \text{if } d_i \le d_0 \\ \frac{A_0}{(d_i/d_0)^\kappa} & \text{if } d_0 < d_i \le d_{max} \\ 0 & \text{if } d_i > d_{max} \end{cases} \tag{3}$$

where; d_i is the distance between the i-th sensor and the target/phenomena, d_{max} is the detection radius and κ is a decay factor that depends on the environment. We note however, that our proposed deployment framework is general and does not depend on any particular choice of the signal amplitude function.

A detection decision is made regarding the presence or absence of a target at the j-th point on the grid by combining the available sensor measurements. This is done at a fusion center (FC), which calculates a decision statistic T_j and compares it to a decision threshold $\eta(j, n_j)$, where n_j is the number of sensors reporting measurements regarding the j-th point (i.e., less than d_{max} meters from the j-th point). A decision is made according to the following non-randomized decision rule,

$$\delta(T_j) = \begin{cases} H_0 \text{ if } T_j \le \eta(j, n_j) \\ H_1 \text{ if } T_j > \eta(j, n_j). \end{cases} \tag{4}$$

In this paper, we adopt the value fusion detection strategy. The decision statistic T_j is given as the average of the measurements reported by the n_j sensors, i.e.,

$$T_j = \frac{1}{n_j}\sum_{i=1}^{n_j} U_i \tag{5}$$

therefore, the detection probability at the j-th point, denoted as $p_d(j, n_j)$, is given as

$$p_d(j, n_j) = Pr(T_j \ge \eta(j, n_j) | \, H_1 \text{ true}) \tag{6}$$

$$= Pr(\frac{1}{n_j}\sum_{i=1}^{n_j}[A(d_i) + N_i] \ge \eta(j, n_j)) \tag{7}$$

$$= Q(\frac{\sqrt{n_j}\eta(j, n_j)}{\sigma} - \frac{1}{\sigma\sqrt{n_j}}\sum_{i=1}^{n_j} A(d_i)) \tag{8}$$

where, $Q(x)$ is given as $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{\frac{-t^2}{2}} dt$. Similarly, the false alarm probability, denoted as $p_f(j, n_j)$, at point (j) is given as

$$p_f(j, n_j) = Pr(T_j \geq \eta(j, n_j)|\, H_0 \text{ true}) \tag{9}$$

$$= Pr(\frac{1}{n_j} \sum_{i=1}^{n_j} N_i \geq \eta(j, n_j)|\, H_0 \text{ true}) \tag{10}$$

$$= Q(\frac{\sqrt{n_j}\eta(j, n_j)}{\sigma}). \tag{11}$$

The FC calculates the decision statistics associated with each point on the grid. The decision threshold is calculated using knowledge of the number of sensors that are within the detection radius, the noise variance and the false alarm requirement associated with every point. In practice, the FC receives measurements from all sensors in the grid and performs a series of sensor measurement averages corresponding to each point on the grid.

The deployment problem that we examine in this work can now be stated as follows: Given $\mathbf{p}_f^{req}$ and $\mathbf{p}_d^{req}$ and a fixed number of sensors K, how can we deploy these sensors in a value fusion based detection system, such that the squared error (SE) between achieved and required detection probabilities is minimized while satisfying false alarm requirements? If we denote the achieved false alarm and detection probability vectors after K sensor have been deployed as $\mathbf{p}_{f,K}$ and $\mathbf{p}_{d,K}$, then we can mathematically state our problem as

$$\underset{\mathbf{u}}{\operatorname{argmin}} \sum_{j:p_{d,K}(j)<p_d^{req}(j)} (\mathbf{p}_{d,K}(j) - \mathbf{p}_d^{req}(j))^2$$

$$\text{subject to} \begin{cases} \mathbf{p}_{f,K} = \mathbf{p}_f^{req} \\ \mathbf{1}^T\mathbf{u} = K \end{cases} \tag{12}$$

where, $\mathbf{u}$ is the deployment vector. The deployment vector is an $N_xN_y \times 1$ vector. Its entries indicate the number of sensors at each point on the grid, and take values of either 0 or 1. $\mathbf{1}^T$ indicates the transpose of an $N_xN_y \times 1$, with all entries set to 1.

3 Optimal Control Formulation

The deployment problem stated earlier can be thought of as a optimal control problem. The SE between achieved and required detection probabilities can be mapped into the cost function to be minimized in an optimal control problem. Furthermore, the set of optimal control vectors correspond to the sensor positions on the grid (i.e., the deployment vector). In an LQR problem, the optimal control vectors are solved sequentially, this means that in the proposed framework sensors are sequentially placed on the grid. In the next section, we illustrate that it is indeed possible to approximate the deployment problem as a linear quadratic regulator (LQR) problem. We will also discuss solving for the optimal control vectors (i.e., deployment vector in our problem) in the LQR problem.

3.1 System Linearization

In an LQR problem, the evolution of the state of the system (i.e., difference in detection probabilities in our problem) is governed by a linear relationship. In this section, we show that it is possible to linearly approximate the change in the difference between $\ln(\frac{1}{1-p_d(j,n_j)} - 1)$ and $\ln(\frac{1}{1-p_d^{req}(j)} - 1)$. Due to the monotone nature of the logarithmic function, minimizing the difference between these two quantities is equivalent to minimizing the difference between $p_d(j,n_j)$ and $p_d^{req}(j)$. Furthermore, we quantify the effect each entry in the control vector will have on the system's evolution.

Noting that we can approximate the $Q(\cdot)$ function as [5]

$$Q(x) \approx 1 - \frac{1}{1+e^{-\sqrt{2}x}}, \tag{13}$$

we can approximate $\ln(\frac{1}{1-p_d(j,n_j)} - 1)$ as follows

$$\ln(\frac{1}{1-p_d(j,n_j)} - 1) \approx \sqrt{\frac{2}{\sigma^2 n_j}} \sum_{i=1}^{n_j} A(d_i) - \sqrt{\frac{2n_j}{\sigma^2}}\eta(j,n_j). \tag{14}$$

Eqn.(14), illustrates that for a fixed n_j and $\eta(j,n_j)$, the change in $\ln(\frac{1}{1-p_d(j,n_j)} - 1)$ is approximately linear with respect to the signal amplitudes measured by the n_j sensors.

Define $m(j,n_j) = \ln(\frac{1}{1-p_d(j,n_j)} - 1)$ and $m^{req}(j) = \ln(\frac{1}{1-p_d^{req}(j)} - 1)$. Having $n_j = k-1$, then it is possible to write $x(j,k) = m(j,k) - m^{req}(j)$ after the k-th sensor has been added within the detection radius of point (j), as follows

$$\begin{aligned} x(j,k) = x(j,k-1) &+ \sqrt{\frac{2}{\sigma^2 k}} A(d_k) \\ &+ \sqrt{\frac{2}{\sigma^2}}(\frac{1}{\sqrt{k}} - \frac{1}{\sqrt{k-1}}) \sum_{i=1}^{n_j-1} A(d_i) \\ &+ \sqrt{\frac{2k}{\sigma^2}}\eta(j,k) + \sqrt{\frac{2(k-1)}{\sigma^2}}\eta(j,k-1). \end{aligned} \tag{15}$$

It is possible to write Eqn.(15) in matrix form as follows

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k. \tag{16}$$

where, $\mathbf{x}_K = [x(j,n_j), \forall j \in \mathcal{G}]^T$. k is the total number of sensors in the grid. We note that, depending on the detection radius, the number of sensor covering a point (j) might be in general less than k. The matrix B is of dimension $N_xN_y \times N_xN_y$. The elements of the B, quantify the contribution of possible sensor positions to the detection probability. For example, the (r,c)th element of $\mathbf{B}$ is given as

$$\begin{aligned}\mathbf{B}(r,c) &= \sqrt{\frac{2}{\sigma^2(n_r+1)}}A(d(r,c)) - \sqrt{\frac{2(n_r+1)}{\sigma^2}}\eta(j,n_r+1) \\ &+ \sqrt{\frac{2}{\sigma^2}}(\frac{1}{\sqrt{n_r+1}} - \frac{1}{\sqrt{n_r}})\sum_{i=1}^{n_r}A(d(r,i)) \\ &+ \sqrt{\frac{2(n_r)}{\sigma^2}}\eta(j,n_r) \end{aligned} \tag{17}$$

where, $d(r,c)$ is the distance between points r and c on the grid. The deployment vector $\mathbf{u}$ is an $N_xN_y \times 1$ vector, with either 0 or 1 entries. The entry value indicates the number of sensors at the point on the grid that corresponds to that entry.

Note that the SE between achieved and required detection probabilities, can be described as the weighted quadratic norm of the state ($\mathbf{x}_k$) of the system described in Eqn.(16). Assuming that the weighted quadratic norm of the system's state is chosen as the cost function and the deployment vector corresponds to a control vector, we are motivated to solve the deployment problem as an optimal control problem. Here, the objective is to determine the control vector that would minimize the cost function. That is, the deployment problem in Eqn.(12) can be restated as

$$\begin{aligned} &\underset{\mathbf{u}_k}{\operatorname{argmin}}\, J = \frac{1}{2}\mathbf{x}_K^T\mathbf{Q}_f\mathbf{x}_K + \frac{1}{2}\sum_{k=1}^{K-1}(\mathbf{x}_k^T\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k) \\ &\text{subject to} \begin{cases} \mathbf{p}_{f,K} = \mathbf{p}_f^{req} \\ \mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{B}_k u_k,\ k = 1,\ldots,K \\ \mathbf{1}^T\mathbf{u} = K \end{cases} \end{aligned} \tag{18}$$

where, $\mathbf{Q}$,$\mathbf{Q}_f$ and $\mathbf{R}$ are symmetric positive definite weighing matrices. The squared error cost function penalizes both positive and negative deviations from the required detection probability profile. To avoid incurring a penalty for satisfying/exceeding detection requirements, the error terms corresponding to a point where the detection requirement has been met/exceeded is set to zero in J. The optimal control problem corresponding to our system is the linear quadratic regulator (LQR) problem. We note here, that the cost function J does not incorporate the false alarm requirements. However, false alarm requirements can be always met by choosing a suitable detection threshold at the FC. In the next section, we discuss the dynamic optimization method of solving the LQR problem.

3.2 Dynamic Optimization

Our system equation corresponds to

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k, \tag{19}$$

and the cost function J that we wish to minimize is given as

$$J = \frac{1}{2}\mathbf{x}_K^T \mathbf{Q}_f \mathbf{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}(\mathbf{x}_k^T \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T \mathbf{R}\mathbf{u}_k). \tag{20}$$

Applying the discrete KKT conditions, results in the following optimality conditions:

$$\begin{align}
\mathbf{x}_0 &= \mathbf{x}_{initial} \tag{21}\\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k \tag{22}\\
\lambda_k &= \lambda_{k+1} + \mathbf{Q}\mathbf{x}_k \tag{23}\\
\lambda_K &= \mathbf{Q}_f \mathbf{x}_K \tag{24}\\
\mathbf{u}_k &= -\mathbf{R}^{-1}\mathbf{B}_k^T \lambda_k \tag{25}
\end{align}$$

The optimal $\mathbf{u}_k$ can be found using the sweep method. In the sweep method, the Lagrange multiplier λ_k is assumed to be given as

$$\lambda_k = \mathbf{P}_k \mathbf{x}_k \tag{26}$$

where, $\mathbf{P}_k$ is a matrix of dimension $N_x N_y \times N_x N_y$. The optimal vector $\mathbf{u}_k$ is then given as:

$$\mathbf{u}_k = -\mathbf{R}^{-1}\mathbf{B}_k^T \lambda_{k+1} \tag{27}$$

but in terms of $\mathbf{x}_k$, Eqn.(26) can be written as follows

$$\begin{align}
\mathbf{u}_k &= -\mathbf{R}^{-1}\mathbf{B}_k^T \mathbf{P}_{k+1}\mathbf{x}_{k+1} \tag{28}\\
&= -\mathbf{R}^{-1}\mathbf{B}_k^T \mathbf{P}_{k+1}\mathbf{x}_k \nonumber\\
&\quad - \mathbf{R}^{-1}\mathbf{B}_k^T \mathbf{P}_{k+1}\mathbf{B}_k \mathbf{u}_k \tag{29}
\end{align}$$

$\mathbf{u}_k$ can now be expressed in terms of $\mathbf{x}_k$ as follows

$$\mathbf{u}_k = -\mathbf{S}^{-1}\mathbf{R}^{-1}\mathbf{B}_k^T \mathbf{P}_{k+1}\mathbf{x}_k \tag{30}$$

where, the matrix $\mathbf{S}$ is of dimension $N_x N_y \times N_x N_y$ and is defined as

$$\mathbf{S} = \mathbf{I} + \mathbf{R}^{-1}\mathbf{B}_k^T \mathbf{P}_{k+1}\mathbf{B}_k \tag{31}$$

In order to have a binary integer solution, a 1 is placed at the index where $\mathbf{u}_k$ is maximum and a 0 is placed at the remaining positions. That is, a sensor is placed at the location corresponding to the index where $\mathbf{u}_k$ is maximum. The resulting vector is denoted as $\mathbf{u}_k^o$. The use of the sweep method involves several matrix operations , with a complexity of $\mathcal{O}(K(3N^3 + 4N^2))$ where $N = N_x N_y$. In the next section we propose a second deployment algorithm that circumvents the need to operate on several matrices.

4 Suboptimal Deployment Algorithm

Our system equation is as follows

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k. \tag{32}$$

Ideally, it is desirable to deploy sensors such that the resulting $\mathbf{x}_k$ is equal to the zero vector (i.e., $\mathbf{x}_k = \mathbf{0}$ implies the detection requirements have been satisfied). Substituting $\mathbf{x}_k = \mathbf{0}$ in Eqn.(32) and solving for $\mathbf{u}_k$, we get the following ;

$$\mathbf{u}_k = -\mathbf{B}_k^{-1} \mathbf{x}_{k-1}. \tag{33}$$

Similar to the optimal control based algorithm, the resulting deployment vector consists of continuous real-valued elements. Therefore, a 1 is placed at the index where $\mathbf{u}_k$ is maximum and a 0 is placed at the remaining positions. Once again, a sensor is placed at the location corresponding to the index where $\mathbf{u}_k$ is maximum. The computational complexity of the suboptimal algorithm is $\mathcal{O}(K(N^3 + N^2))$, where N is as defined earlier. Furthermore, simulation results show that the performance of this algorithm is comparable to that of the optimal control based algorithm.

5 Simulation Results

In this section, we compare the performance of the greedy, suboptimal and optimal control based algorithms. In the greedy algorithm, a sensor is placed at the point with the largest difference between required and achieved detection probability.

In the first experiment, the area of interest is modeled as a grid of 25×25 points. The false alarm and detection probability are uniform and are set to $p_f^{req} = 0.01$ and $p_d^{req} = 0.9$, respectively. The noise variance is set to $\sigma^2 = 1$ and $d_0 = 1$. A discussion of the choice of d_{max} can be found in [4]. Table 1, lists the number of sensors needed by each algorithm, to meet the false alarm and detection requirements as the initial signal amplitude and detection radius are varied. The minimum numbers of sensors can be found by assuming, in the problem statement and LQR formulation, a large number of sensors K, and deploying sensors till detection and false alarm requirements are met. Results in Table 1 indicate, that the optimal control based algorithm uses 25% fewer sensors than the greedy algorithm. This is due to the fact, that in the greedy algorithm a sensor is deployed by anticipating the effect the sensor deployment will have at a single point (i.e., the point with the largest difference between required and achieved detection probabilities). In contrast, in the proposed algorithms, the deployment process takes into account the effect of each sensor deployment on the whole grid, which is embedded in the matrix $\mathbf{B}$. Fig. 1 shows the convergence of the SE between required and achieved detection probabilities as a function of the number of sensors deployed in the network by each algorithm, for the case of $A_0 = 100$ and $d_{max} = 5.26$ meters. We note that the suboptimal and

Table 1. Number of sensors for uniform requirements

Parameters	Greedy	Sub-optimal	Optimal
$A_0 = 30, d_{max} = 2.9$	58	56	53
$A_0 = 50, d_{max} = 3.72$	36	35	30
$A_0 = 100, d_{max} = 5.26$	20	16	15
$A_0 = 250, d_{max} = 8.3$	9	8	8

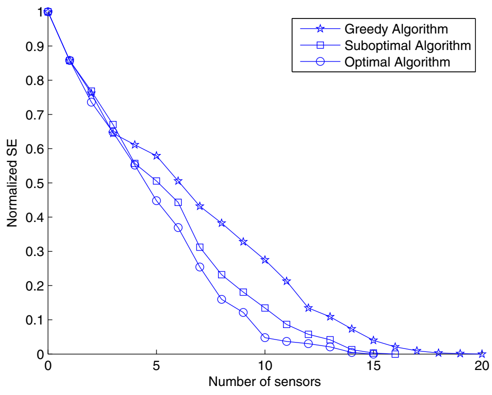


Fig. 1. SE convergence for uniform requirements

optimal control based algorithms have a faster convergence rate than the greedy algorithm.

In the second experiment, we consider a setup similar to the first experiment with a fixed initial amplitude of $A_0 = 50$. However, the false alarm and detection requirements are not uniform over the grid (see Fig. 2). In Table 2, the number of sensors needed by each algorithm to satisfy the same $\mathbf{p}_d^{req}$ and $\mathbf{p}_f^{req}$ requirements is indicated. Results indicate that even with nonuniform requirements, the proposed algorithm uses fewer number of sensors than the greedy algorithm.

Table 2. Number of sensors for uniform requirements

Requirements	Greedy	Suboptimal	Optimal
$p_{d1} = 0.9, p_{d2} = 0.7, p_{f1} = 0.01, p_{f2} = 0.001$	35	33	31
$p_{d1} = 0.9, p_{d2} = 0.7, p_{f1} = 0.001, p_{f2} = 0.01$	30	27	26
$p_{d1} = 0.9, p_{d2} = 0.9, p_{f1} = 0.01, p_{f2} = 0.001$	40	37	34

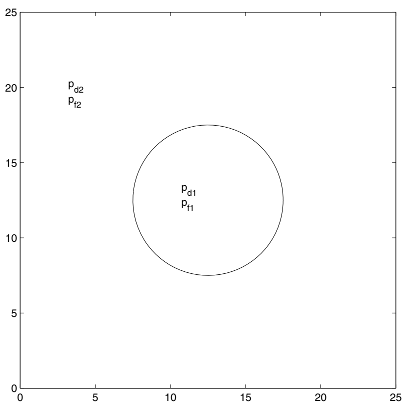


Fig. 2. Nonuniform requirements

6 Conclusion

In this paper, we study the sensor deployment problem in a distributed sensor network employing value fusion. Our objective is to minimize the SE between achieved and required detection probabilities while not exceeding false alarm requirements. We propose modeling the problem as a linear quadratic regulator (LQR) problem. Sensors are sequentially deployed by solving for the control vectors in the LQR problem. We also propose a suboptimal deployment algorithm with lower computational complexity than the optimal control based algorithm. Simulation results indicate that the proposed algorithms have a faster SE convergence rate in comparison to a greedy algorithm. That is, the proposed algorithms use fewer sensors than the greedy algorithm to satisfy the same detection and false alarm requirements requirements.

References

1. Zou, Y., Chakrabarty, K.: Uncertainty-aware and coverage-oriented deployment for sensor networks. Journal of Parallel and Distributed Computing 64(7), 788–798 (2004)
2. Zhang, J., Yan, T., Son, S.H.: Deployment strategies for differentiated detection in wireless sensor networks. In: 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, SECON 2006, September 28, vol. 1, pp. 316–325 (2006)
3. Viswanathan, R., Varshney, P.: Distributed detection with multiple sensors i. fundamentals. Proceedings of the IEEE 85(1), 54–63 (1997)
4. Yuan, Z., Tan, R., Xing, G., Lu, C., Chen, Y., Wang, J.: Fast sensor deployment for fusion-based target detection. In: IEEE Real-Time Systems Symposium (RTSS 2008) (December 2008)
5. Raykar, V.C., Duraiswami, R., Krishnapuram, B.: A fast algorithm for learning a ranking function from large-scale data sets. IEEE Transactions on Pattern Analysis and Machine Intelligence 3(7), 1158–1170 (2008)

Approximate Distributed Kalman Filtering for Cooperative Multi-agent Localization

Prabir Barooah[1], Wm. Joshua Russell[2], and João P. Hespanha[2,⋆]

[1] University of Florida, Gainesville, FL 32611, USA
pbarooah@ufl.edu
[2] University of California, Santa Barbara, CA 93106, USA
wjrussell@umail.ucsb.edu, hespanha@ece.ucsb.edu

Abstract. We consider the problem of estimating the locations of mobile agents by fusing the measurements of displacements of the agents as well as relative position measurements between pairs of agents. We propose an algorithm that computes an approximation of the centralized optimal (Kalman filter) estimates. The algorithm is distributed in the sense each agent can estimate its own position by communication only with nearby agents. The problem of distributed Kalman filtering for this application is reformulated as a parameter estimation problem. The graph structure underlying the reformulated problem makes it computable in a distributed manner using iterative methods of solving linear equations. With finite memory and limited number of iterations before new measurements are obtained, the algorithm produces an approximation of the Kalman filter estimates. As the memory of each agent and the number of iterations between each time step are increased, the approximation improves. Simulations are presented that show that even with small memory size and few iterations, the estimates are quite close to the centralized optimal. The error covariances of the location estimates produced by the proposed algorithm are significantly lower than what is possible if inter-agent relative position measurements are not available.

1 Introduction

Mobile autonomous agents such as unmanned ground robots and unmanned aerial vehicles that are equipped with on-board sensing, actuation, computation and communication capabilities hold great promise for applications such as surveillance, disaster relief, and scientific exploration. Irrespective of the application, their successful use generally requires that the agents be able to obtain accurate estimates of their positions. Although typically position information is provided by GPS, in many scenarios GPS may be available only intermittently, or sometimes not available at all. These situations include underwater operation,

⋆ The research reported in this paper is based upon work supported by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D-0004 from the U.S. Army Research Office and by the National Science Foundation under Grant No. CCR-0311084.

presence of urban canyons, or hostile jamming. In such a situation, localization is typically performed by integrating measurements of displacements, which can be obtained from IMUs (inertial measurement units) or/and vision sensors [1–3]. Since these measurements are noisy, their integration over time lead to high rate of error growth [3–5].

When multiple agents operate cooperatively, it is possible to reduce localization errors if information on *relative positions* between pairs of agents are available. Such measurements can be obtained by vision-based sensors such as cameras and LIDARs, or RF sensors through AoA and TDoA measurements. These measurements, although noisy, furnish information about the agent's locations in addition to that provided by displacement measurements. The problem of fusing information on relative positions between mobile agents for estimating their locations is commonly known as *cooperative localization* in the robotics literature [6–8]. The typical approach is to use an extended Kalman filter, to fuse both odometry data and robot-to-robot relative distance measurements [8, 9].

However, computing the Kalman filter estimates requires that all the measurements (inter-agent as well as displacement measurements of all the agents) are made available to a central processor. Such a centralized approach requires routing data though an ad-hoc network of mobile agents, which is a difficult problem [10]. Therefore, a *distributed scheme* that allows agents to estimate their positions accurately through local computation and communication instead of relying on a central processor is preferable. In the sensor networks and control literature, the problem of distributed Kalman filtering has drawn quite a bit of attention [11–14]. The papers [12, 15] in particular present a distributed Kalman filtering techniques for multi-agent localization. The paper [16] addresses cooperative localization in mobile sensor networks with intermittent communication, in which an agent updates its prediction based on the agents it encounters, but does not use inter-agent relative position measurements.

In this paper we approach the problem of distributed Kalman filtering for cooperative localization from a novel angle, by reformulating the problem as a parameter estimation problem. The Kalman filter computes the LMMSE estimates of the states of a linear system given the measurements. It is a classical result that under infinite prior covariance, the LMMSE is the same as the BLUE (best linear unbiased estimator) [17]. For the problem at hand, the BLUE estimator has a convenient structure that can be described in terms of a graph consisting of nodes (agent locations) and edges (relative measurements). This structure can be exploited to distribute the computations by using parallel iterative methods of solving linear equations. The proposed method therefore is designed to compute the BLUE estimates using iterative techniques, and is based our earlier work on localization in static sensor networks [18, 19]. If the agents were to have infinite memory and could run infinitely many iterations before they move and change their positions, the estimates produced by the proposed algorithm are equal to the centralized BLUE estimates, and therefore the same as the Kalman filter estimates. Due to memory and time constraints, the proposed algorithm uses only a small subset of all the past measurements and runs only one (or a few) iterations. This makes the method an approximation of the Kalman filter. Fewer the number of iterations and smaller the size of past measurements retained, the easier it is to implement the algorithm in a distributed setting. Simulations show the approximations are remarkably close to the centralized Kalman filter/BLUE estimates even when a very small amount of past data is used and only one iteration is executed between successive motion updates.

The rest of the paper is organized as follows. Section 2 describes the estimation problem precisely. Section 3 describes centralized Kalman filtering for cooperative localization and its reformulation as a problem of parameter estimation in measurement graphs. Section 4 describes the proposed algorithm, and simulations are presented in Section 5.

2 Problem Description

Consider a group of n mobile agents that need to estimate their own positions with respect to a geostationary coordinate frame, whose origin is denoted by x_0. In the absence of GPS, we arbitrarily fix the initial position of one of the agents, say, the first agent, as the origin. Time is measured with a discrete index $k = 0, 1, 2, \ldots$ The noisy measurement of the displacement of agent j obtained by its on-board sensors during the k-th time interval is denoted by $u_j(k)$, so that

$$u_j(k) = x_j(k+1) - x_j(k) - w_j(k), \tag{1}$$

where $x_j(k)$ is the position of agent j at time k, and $\{w_j(k)\}$ is a zero-mean noise with the property that $E[w_j(k)w_i(\ell)] = 0$ unless $i = j, k = \ell$. We assume that certain pairs of agents, say i, j, can also obtain noisy measurements $y_{ij}(k)$ of their relative position

$$y_{ij}(k) = x_i(k) - x_j(k) + v_{ij}(k), \tag{2}$$

where $v_{ij}(k)$ is zero-mean measurement noise with $E[v_{ij}(k)v_{mn}(\ell)] = 0$ unless $(ij) = (mn), k = \ell$. We assume that the agents are equipped with compasses, so that all these measurements are expressed in a common Cartesian reference frame. The goal is to combine the agent-to-agent relative position measurements with agent displacement measurements to obtain estimates of their locations that are more accurate than what is possible from the agent displacement measurements alone. In addition, the computation should be distributed so that every agent can compute its own position estimate by communication with a small subset of the other agents, called *neighbors*. We assume that two agents that can obtain each others' relative position measurement at time index k can also exchange information through wireless communication during the interval from k to $k+1$.

3 Kalman Filtering vs. BLU Estimation from Relative Measurements

The availability of the displacement measurements allow us to write the following process model for the j-th vehicle:

$$x_j(k+1) = x_j(k) + u_j(k) + w_j(k), \tag{3}$$

where $u_j(k)$ is now viewed as a known input. One can stack the states $x_j(k), j = 1, \ldots, n$ into a tall vector $\mathbf{x}(k)$ and write the system dynamics

$$\mathbf{x}(k+1) = \mathbf{x}_j(k) + \mathbf{u}(k) + \mathbf{w}(k), \quad \mathbf{y}(k) = C(k)\mathbf{x}(k) + \mathbf{v}(k)$$

where $C(k)$ is appropriately defined so that entries of $\mathbf{y}(k)$ are the inter-agent relative position measurements (2). When the control input and the measurements $\{\mathbf{u}(t)\}, \{\mathbf{y}(t)\}, t \in \{0, 1, \ldots, k\}$ are made available to a central processor, along with the initial conditions $\hat{\mathbf{x}}(0|-1)$ and $P(0|-1) = Cov(\hat{\mathbf{x}}(0|-1) - \mathbf{x}(0), \hat{\mathbf{x}}(0|-1) - \mathbf{x}(0))$, and the noise covariances $Q(t) := Cov(\mathbf{w}(t), \mathbf{w}(t))$, $R(t) := Cov(\mathbf{v}(t), \mathbf{v}(t))$, a Kalman filter can be used to compute estimate $\hat{\mathbf{x}}^{\text{Kalman}}(k|k) = E * (\hat{x}(k)|\{u\}, \{y\})$ of the state $\mathbf{x}(t)$, where $E * (X|Y)$ denotes the LMMSE estimate of a r.v. X in terms of the r.v. Y [20].

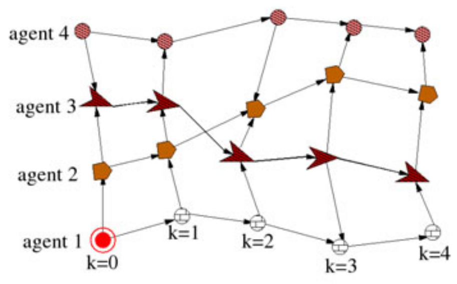
To distribute the computations of the Kalman filter, we reformulate the problem into an equivalent, deterministic parameter estimation problem. We associate the positions $\{x_0\} \cup \{x_j(t)\}$, $j \in \{1, \ldots, n\}$, $t \in \{0, 1, \ldots, k\}$ of the entities until time k with the *nodes* $\mathbf{V}(k)$ of a *measurement graph* $\mathbf{G}(k) = (\mathbf{V}(k), \mathbf{E}(k))$. The *edges* $\mathbf{E}(k)$ of the graph correspond to the relative position measurements between the nodes $\mathbf{V}(k)$. If an edge is between two nodes that correspond to subsequent positions of same agent j, then the edge corresponds to a measurement of the displacement of the agent between those two time instants. If , on the other hand, the edge is between two nodes that correspond to the positions of two distinct agents at a particular time instants, then the edge corresponds to the noisy measurement of the relative position between the agents. All measurements mentioned above are of the type

$$\zeta_e = x_u - x_v + \epsilon_e \tag{4}$$

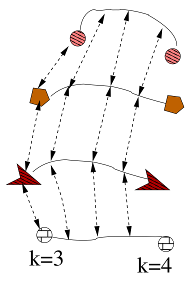
where u and v are nodes of the measurement graph $\mathbf{G}(k)$ and $e = (u, v)$ is an edge (an ordered pair of nodes). In particular, an edge exists between a node pair u and v if and only if a relative measurement of the form (4) is available between the two nodes. Since a measurement of $x_u - x_v$ is different from that of $x_v - x_u$, the edges in the measurement graph are directed. The edge directions are arbitrary. Figure 1 shows an example of a measurement graph.

3.1 BLUE Estimation

We briefly review the BLUE (best linear unbiased estimator) from relative measurements for graphs that do not change with time. The BLUE is optimal (minimal variance) among all linear unbiased estimators. Consider a measurement



(a) Measurement graph $\mathbf{G}(4)$.



(b) Communication between $k = 3$ and $k = 4$.

Fig. 1. (a) An example of a measurement graph generated as a result of the motion of a group of four mobile agents. The graph shown here is $\mathbf{G}(4)$, i.e., the snapshot at the 4th time instant. The unknown variables at current time $k = 4$ are the positions $x_i(t)$, $i \in \{1, 2, \ldots, 4\}$, at the time instants $k \in \{0, 1, \ldots, 4\}$, except for the initial position of agent 1: $x_1(0)$, which is taken as the reference. (b) The communication during the time interval between $k = 3$ and $k = 4$. In the situations shown in the figure, 4 rounds of communication occur between a pair of agents in this time interval.

graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where the nodes in $\mathbf{V}$ correspond to variables and edges in $\mathbf{E}$ correspond to relative measurement between node variables of the form (4). Let $V_r \subset \mathbf{V}$ denote the non-empty subset of nodes whose variables are known, which are called *reference variables* , and $n = |\mathbf{V} \setminus \mathbf{V}_r|$ be the number of unknown variables that are to be estimated. Let $\mathbf{x}$ be the vector obtained by stacking together the unknown variables. As described in [19], given a measurement graph with n unknown variables, the BLU estimate $\hat{\mathbf{x}}^*$ is given by the solution of a system of linear equations

$$\mathcal{L}\mathbf{x} = \mathbf{b}, \tag{5}$$

where $\mathcal{L}$ and $\mathbf{b}$ depend on the measurement graph $\mathbf{G}$, the measurement error covariance matrices $P_e, e \in \mathbf{E}$, the measurements $\zeta_e, e \in \mathbf{E}$ and the reference variables $x_r, r \in \mathbf{V}_r$. The matrix $\mathcal{L}$ is invertible (so that BLU estimate $\hat{\mathbf{x}}^*$ exists and is unique) if and only if for every node, there is an undirected path between the node and at least one reference node [21]. Under this condition, the covariance matrix of the estimation error $\Sigma := cov(\hat{\mathbf{x}}^*, \hat{\mathbf{x}}^*)$ is given by

$$\Sigma = \mathcal{L}^{-1}. \tag{6}$$

By splitting the matrix $\mathcal{L} = M - N$ where M is a block diagonal degree matrix and N is a generalized adjacency matrix of the network, the above system of equations can be written as $M\mathbf{x} = N\mathbf{x} + \mathbf{b}$, which leads to the following block-Jacobi iterative method for solving it:

$$\hat{\mathbf{x}}(k+1) = M^{-1}N\hat{\mathbf{x}}(k) + M^{-1}\mathbf{b}. \tag{7}$$

where M is block diagonal and N is sparse. In particular, only those entries on the i-th block row of N are non-zero that corresponds to i's neighbors in $\mathbf{G}$. This structure makes it possible to compute the updates in (7) in a distributed manner, so that each node only needs to communicate with its neighbors. The resulting algorithm can be shown to converge under certain assumptions on the measurement error covariance matrices, even when executed in an asynchronous manner, and in the presence of temporary communication faults. The reader is referred to [19, 21] for the details.

When the measurement graph is time-varying, if all the measurements corresponding to the edges in $\mathbf{G}(k)$ are available to a central processor at time k, the processor can compute the BLU estimates of all the node variables in the graph $\mathbf{G}(k)$ (which correspond to the present as well as past positions of the agents) by solving (5). The resulting estimate is denoted by $\hat{\mathbf{x}}^{\text{BLUE}}(k)$.

The following result, which follows from standard results in estimation theory shows that under uninformative prior, the blue estimate is equivalent to the Kalman filter estimates.

Lemma 1. *If* $P^{-1}(0|-1) = 0$*, then* $\hat{x}^{\text{Kalman}}(k|k) = \hat{\mathbf{x}}^{\text{BLUE}}(k)$ *for every* k. □

For the problem at hand, we assume that no prior information on the agent positions are available at time 0 except for inter-agent position measurements $\mathbf{y}(0)$ and the position of one of the agents that is used as a reference. In that case the information form of the Kalman filter can be used to compute the LMMSE estimate of the agent positions [22]. The result above shows that under the assumption of no prior information, the BLUE estimates are identical to the Kalman filter estimates. Therefore, from now on we do not distinguish between $\hat{x}^{\text{Kalman}}(k|k)$ and $\hat{\mathbf{x}}^{\text{BLUE}}(k)$, and refer to them simply as the centralized optimal estimate $\hat{\mathbf{x}}^{*}(k)$.

In the next section we will utilize the BLUE formulation to devise distributed algorithms to compute the estimates.

4 A Distributed Algorithm for Dynamic Localization

In this section we present a distributed algorithm to obtain estimates of the positions mobile agents that are close to the centralized BLU estimator described in the previous section. By distributed we mean every agent should be able to estimate its own position, and all the information needed to carry out the computation should come from local sensing and communication with its neighboring agents.

4.1 Infinite Memory and Bandwidth

We first describe the algorithm by assuming that every agent can store and broadcast an unbounded amount of data. We will relax this assumption later.

For every agent j, let $\mathbf{V}_j(k)$ contain all the nodes that correspond to the positions of itself and the positions of the agents with whom j has had relative

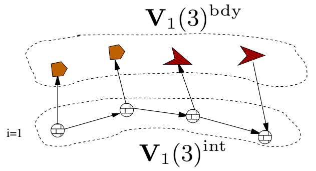

$\mathbf{V}_1(3)^{\text{bdy}}$
i=1
$\mathbf{V}_1(3)^{\text{int}}$

Fig. 2. The subgraph $\mathbf{G}_1(3)$ of agent 1 at time 3, for the measurement graph shown in Figure 1

measurements, up to and including time k. Let $\mathbf{E}_j(k)$ be the subset of edges in $\mathbf{G}(k)$ that are incident on nodes that correspond to j's current or past positions. By assumption, the relative measurements and their error covariances $\zeta_e, P_e, e \in \mathbf{E}_j(k)$ are available to agent j at time k. We now define the *local subgraph* of agent j at time k as $\mathbf{G}_j(k) = (\mathbf{V}_j(k), \mathbf{E}_j(k))$. Figure 2 shows the subgraph of agents 1 at time $k = 3$ corresponding to the measurement graph shown in Figure 1.

The nodes in a local subgraph $\mathbf{G}_j(k)$ are divided into two categories - the *internal nodes* $\mathbf{V}_j(k)^{\text{int}}$ and the *boundary nodes* $\mathbf{V}_j(k)^{\text{bdy}}$. The internal nodes are the nodes that correspond to the positions of the agent up and including time t. The boundary nodes consist of the nodes in the local subgraph that correspond to the positions of the neighboring agents. Thus, $\mathbf{V}_j(k) = \mathbf{V}_j(k)^{\text{int}} \cup \mathbf{V}_j(k)^{\text{bdy}}$ and $\mathbf{V}_j(k)^{\text{int}} \cap \mathbf{V}_j(k)^{\text{bdy}} = \emptyset$.

To explain the algorithm, imagine first that the agents have stopped moving at time t_{stop}. We have proposed a distributed algorithm in [19, 23] for localization of static agents that is based on the Jacobi iterative method of solving linear equations [24]. If agents stop moving, they can use the Jacobi algorithm to compute the optimal estimate of its entire position history in distributed manner. We describe the procedure briefly, which will serve as a stepping stone into developing the proposed algorithm.

Algorithm A.1: static agents. Let o be the global reference node. Consider the set of past positions of agent j until time t_{stop}, i.e., $\{x_v, v \in \mathbf{V}_j(t_{\text{stop}})^{\text{int}} \setminus \{o\}\}$. The vector of the node variables in this set is denoted by $\mathbf{x}_j(t_{\text{stop}})$. The set of unknown node positions at time t_{stop} is $\cup_j \mathbf{x}_j(t_{\text{stop}})$. Let $\hat{\mathbf{x}}_j^\tau(t_{\text{stop}})$ be the estimate of $\mathbf{x}_j(t_{\text{stop}})$ obtained by agent j at the end of the τ^{th} iteration. The estimates are obtained and improved using the following distributed algorithm, starting with an arbitrary initial condition:

At τ^{th} iteration, every agent j does the following.

1. It broadcasts the current estimate $\hat{\mathbf{x}}_j^\tau(t_{\text{stop}})$ to all of its neighboring agents. Consequently, it also receives the current estimates $\hat{\mathbf{x}}_i^\tau(t_{\text{stop}})$ from each of its neighboring agents, i.
2. It (agent j) assigns the boundary nodes $\mathbf{V}_j(t_{\text{stop}})^{\text{bdy}}$ as the reference nodes of its local subgraph $\mathbf{G}_j(t_{\text{stop}})$ and sets the reference variables to be the estimates of those node variables that it has recently received from its neighbors. With this assignment of reference node variables and with the relative

measurements $\{\zeta_e, e \in \mathbf{E}_j(t_{\text{stop}})\}$, agent j then sets up the system of linear equations (5) for its local subgraph $\mathbf{G}_j(t_{\text{stop}})$, and solves these equations to obtain an updated estimate of $\hat{\mathbf{x}}_j^{\tau+1}(t_{\text{stop}})$ of its "internal" node variables. $\square$

The following result about the behavior of the estimates follows from the convergence property of the Jacobi algorithm (see [19, 23]).

Proposition 1. *The estimates of all the node variables* $\mathbf{x}(t_{\text{stop}})$ *(i.e., all agents' past positions up to time* t_{stop}*) converge to their centralized optimal estimates:* $\hat{\mathbf{x}}_j^{\tau}(t_{\text{stop}}) \to \hat{\mathbf{x}}_j^{*}(t_{\text{stop}})$ *as* $\tau \to \infty$. $\square$

As a result, if agents stop moving, by communicating with its neighbors and updating sufficiently many times, an agent can obtain an estimate of its entire position history that is arbitrarily close to the optimal estimates. Note that the description above implicitly assumes that the iterations are executed synchronously, i.e., there is a common iteration counter τ among all the agents. However, the result in Proposition 1 holds even if communication and computation is performed in an asynchronous way, where every agent has its own iteration counter τ^i. This follows from standard results in asynchronous iterations [23].

Now we are ready to describe the algorithm for localization of mobile agents with finite memory and finite bandwidth.

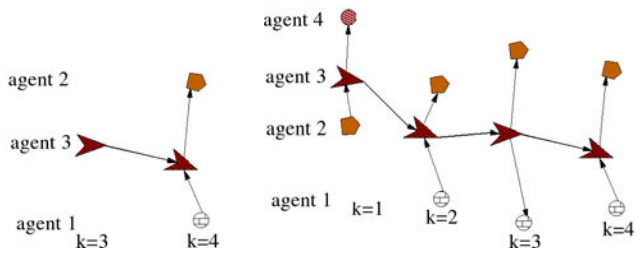
Estimation with mobile agents: Algo A.2. In the description below, T_{mem} is a fixed positive integer that denotes the "size" of a subgraph of $\mathbf{G}(k)$ every agent maintains in its local memory at time k. The parameter T_{mem} is fixed ahead of time and provided to all agents; its value is determined by the memory in each agent's local processor. The maximum number of iterations carried out by an agent j during the interval between times k and $k+1$, which is denoted by $\tau^{\max}$, depends on the maximum number of communication rounds between j and its neighbors during this interval.

Let $\mathbf{G}(k)^{T_{\text{mem}}} = (\mathbf{V}(k)^{T_{\text{mem}}}, \mathbf{E}(k)^{T_{\text{mem}}})$ be the subgraph containing nodes, $\mathbf{V}(t)^{T_{\text{mem}}}$, that correspond to all agent positions from time $\max(k - T_{\text{mem}}, 0)$ until time k and containing edges, $\mathbf{E}(k)^{T_{\text{mem}}}$, corresponding to relative measurements between to nodes in $\mathbf{G}(k)^{T_{\text{mem}}}$. More simply, $\mathbf{G}(k)^{T_{\text{mem}}}$ is the subgraph containing all nodes and edges corresponding to positions of agents and relative measurements at the current and previous T_{mem} time instants. In this case, $\hat{\mathbf{x}}_j^{\tau}(k)$ is a vector of the estimates of the positions of agent j from time instant $\max(k - T_{\text{mem}}, 0)$ to time k, obtained in the τ^{th} iteration.

1. If GPS is not available to every agent at $k = 0$, one agent's initial position serves as the global reference. Every other agent starts with the initial estimate that is obtained by adding the relative measurements on a path from itself to the agent whose initial position is taken as the global reference. For example, when agent 1 is the global reference and relative position measurements are available between agents with successively increasing indices, we have $\hat{x}_j(0) := y_{j,j-1}(0) + y_{j-1,j-2}(0) + \cdots + y_{2,1}(0) + x_1(0)$. We assume that

these measurements are transmitted to the agents initially before they start moving.

2. During the time interval between time indices k and $k+1$, each agent j updates the estimate of $\mathbf{x}_j(t)$ in the following way.
   – initialization: $\hat{x}_j^{(0)}(k) = \hat{x}_j^{(\tau_{\max})}(k-1) + u_j(k-1)$.
   – collect inter-agent measurements, i.e., obtain $\zeta_{v,w}$ for $v = i(k)$ and $w \in \mathcal{N}_j(t)$.
   – iterative update: node j now iteratively updates its position by the algorithm described in the previous section. Specifically, it runs the algorithm A.1 for the subgraph $\mathbf{G}_j(k)^{T_{\text{mem}}}$. □



(a) $T_{\text{mem}} = 1$ (b) $T_{\text{mem}} = 3$

Fig. 3. Truncated subgraphs, $\mathbf{G}_3(4)$ of agent 3 at time 4 for the measurement graph shown in Figure 1

The algorithm continues as long as the agents continue to move. Figure 3 shows an example of the local subgraphs used by an agent (agent 3 in Figure 1) for two cases, $T_{\text{mem}} = 1$ and $T_{\text{mem}} = 3$. Note that the proposed algorithm is particularly simple when $T_{\text{mem}} = 1$, since in that case the iterative update is the solution to the following equation:

$$S_i(k-1)\hat{x}_j^{\tau}(k) = W_j^{-1}(k-1)\Big(\hat{x}_j^{\tau_{\max}}(k-1) + u_j(k-1)\Big) + \sum_{i\in\mathcal{N}_j(k)} V_{j,i}^{-1}(k)\Big(\hat{x}_i^{\tau-1}(k) + y_{j,i}(k)\Big),$$

where $W_j(k) := cov(w_j(k), w_j^T(k))$ and $V_{j,i}(k) := cov(v_{ij}(k), v_{ij}^T(k))$ are the error covariances in the displacement measurement $u_j(k-1)$ and inter-agent relative measurement $y_{ij}(k)$, respectively, and $S_i(k) := W_j^{-1}(k) + \sum_{i\in\mathcal{N}_j(k)} V_{i,j}^{-1}(k)$. When all the measurement error covariances are equal, the update is simply:

$$\hat{x}_j^{\tau}(k) = \frac{1}{\mid N_j(k)+1 \mid}\big(\hat{x}_j^{\tau_{\max}}(k-1) + u_j(k-1) + \sum_{i\in\mathcal{N}_j(k)} (\hat{x}_i^{\tau-1}(k) + y_{j,i}(k))\big)$$

When $T_{\text{mem}} = \infty$, the algorithm is simply the Jacobi iterations to compute the BLUE estimates of all the node variables in the graph $\mathbf{G}(k)$, i.e., the past and

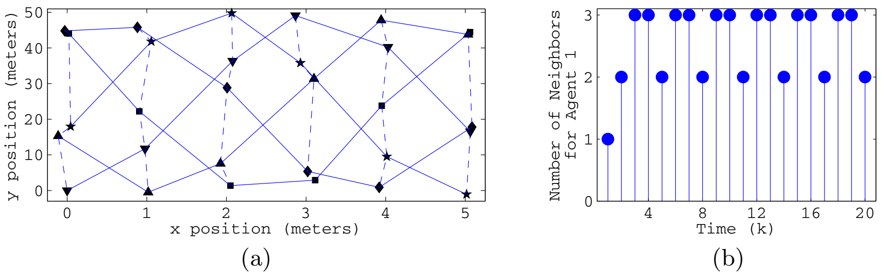


Fig. 4. A snapshot of the measurement graph $\mathbf{G}(k)$ at time $k = 5$ created by the motion of 5 mobile agents, for which the simulations reported here are conducted

present positions of the agents. In this case, Proposition 1 guarantees that the estimates computed converge to the BLUE estimates as $\tau^{\max} \to \infty$. When the algorithm is implemented with small values of T_{mem}, after a certain number of time steps, measurements from times earlier than T_{mem} steps into the past are no longer directly used. Past measurements are still used indirectly, since they affect the values of the reference variables used by the agents for their local subgraphs. With finite T_{mem}, the estimates are no longer guaranteed to reach their centralized optimal. A further reduction in accuracy comes from the fact that in practice $\tau^{\max}$ may not be large enough to get close to convergence even in the truncated local subgraphs. The algorithm therefore produces an approximation of the centralized optimal estimates; the approximation becomes more accurate as $\tau^{\max}$ and T_{mem} increases.

Communication and computation cost. The amount of data an agent needs to store and broadcast increases as the "size" of the truncated local subgraph that the agent keeps in local memory increases, and therefore, on T_{mem}. When the neighbors of an agent do not change with time, the number of nodes in its local truncated subgraph of an agent at any given time is $T_{\text{mem}} + N_{\text{nbr}}T_{\text{mem}}$, where N_{nbr} is the number of neighbors of the agent. In this case, the number of edges in the truncated local subgraph is at most $T_{\text{mem}} + T_{\text{mem}}N_{\text{nbr}}$ (the first term is the number of odometry measurements and the second term is the number of relative measurements between the agent and its neighboring agents that appear as edges in the subgraph). Therefore, an agent needs a local memory large enough to store $\ell[2T_{\text{mem}}(1 + N_{\text{nbr}}) + T_{\text{mem}}N_{\text{nbr}}]$ floating-point numbers, where $\ell = 2$ or 3 depending on whether positions are 2 or 3 dimensional vectors. An agent has to broadcast the current estimates of its interior node variables, i.e., ℓT_{mem} numbers, at every iteration. Thus, the communication, computation and memory requirements of the algorithm are quite low for small values of T_{mem}. We assume that the agents can obtain the error covariances of the measurements on the edges that are incident on themselves, so these need not be transmitted.

5 Simulations

We illustrate the algorithm's performance by numerical simulations. All simulation results are shown for the case $T_{\text{mem}} = 1$. Five agents move in a zig-zag trajectory; the resulting measurement graph is shown in Figure 4(a). A time trace of the number of neighbors of agent 1 is shown in Figure 4(b). The initial position of agent 1 (bottom left node in Figure 4) is taken as the reference. Every measurement of $x_u - x_v$ is obtained from noisy measurements of the distance $\|x_u - x_v\|$ and the angle between x_u and x_v. The distance and angle measurements are corrupted with additive Gaussian noise, with $\sigma_r = 0.05m$ and $\sigma_\theta = 5^o$. The measurement error covariances are estimated from the range and bearing measurements and the parameters σ_r, σ_θ (as explained in [19]), which makes the covariances of the errors on relative position measurements on distinct edges distinct.

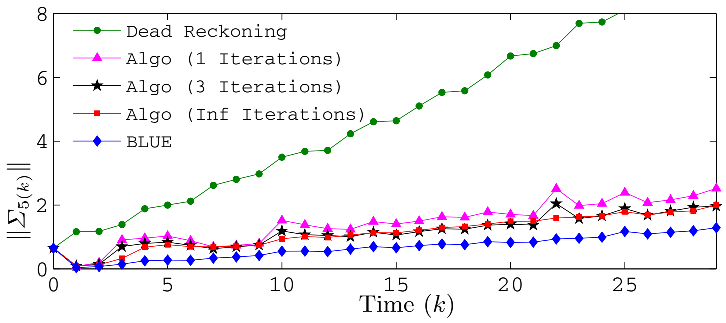


Fig. 5. Covariance of the estimate of the current position of agent 5 (of Figure 4) as a function of time. Agent 5 is the one farthest from agent 1, whose initial position being the reference node. Dead reckoning provides an estimate of positions by summing optometry data. Estimates from algorithm 2 are shown for $\tau^{\max} = 1, 3$, and ∞. BLUE refers to the centralized optimal.

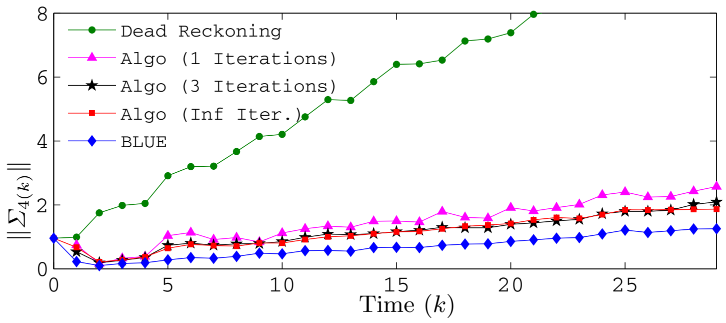


Fig. 6. Covariance of the estimate of the current position of agent 4 (of Figure 4) as a function of time

Covariances of agent position estimates produced by the proposed algorithm are estimated from 1000 Monte-Carlo runs. Figure 5 shows the covariance of the estimate of $x_5(t)$, the position of agent 5, as a function of t. Agent 5 is the one farthest away from agent 1. The figure shows that the location estimates produced by the proposed algorithm are much more accurate than those produced by integrating the displacement measurements alone (dead reckoning). It is seen from the plot that the estimation error covariance of the algorithm (even with $T_{\text{mem}} = 1$ and $\tau^{\max} = 1$) is close to that of the centralized optimal estimator (BLUE). Comparison among the plots for $\tau^{\max} = 1, 3$ and ∞ shows that, as expected, the estimation accuracy improves with increasing number of iterations between every time step. However, it is also seen that the improvement levels off quickly. In fact, even with the minimal possible number of iterations $\tau^{\max} = 1$, the estimation accuracy is quite close to the best possible (with $\tau^{\max} = \infty$). This property of the algorithm enhances its applicability since good estimates are obtained with little delay. Figure 6 plots these variables for the second agent.

6 Conclusion

We presented a distributed algorithm for mobile agents to estimate their positions by fusing their own displacement measurements with inter-agent relative position measurements. The algorithm is distributed in the sense each agent can estimate its own position by communication only with nearby agents. The algorithm computes an approximation of the centralized optimal (Kalman filter) estimates. The problem of distributed Kalman filtering for this application is reformulated as a problem of computing the BLUE estimates. The graph structure of the BLUE estimation problem, which makes it computable using iterative methods of solving linear equations, makes distributing the computations possible. With finite memory and limited number of iterations before new measurements are obtained, the algorithm produces an approximation of the Kalman filter estimates. As the memory of each agent and the number of iterations between each time step are increased, the approximation improves. Simulations show, however, that even with small memory size and a single iteration, the estimates are quite close to the centralized optimal. Simulations further show that the error covariances of the state estimates that the proposed distributed algorithm yield are significantly lower than what is possible by dead reckoning.

There are several aspects of the proposed algorithm that need further investigation. Although numerical simulations show that the estimates produced by the algorithm are close to the centralized optimal estimates, a precise characterization of the difference is lacking. In particular, it will be useful to understand the affect of the parameters T_{mem} and $\tau^{\max}$ on the performance of the algorithm. Moreover, the evolution error covariance will depend on the number of agents and the measurement graph, which is determined by agents' motion. The relationship between the covariance and agent motion is a subject of future research.

References

[1] Borenstein, J., Everett, H.R., Feng, L., Wehe, D.: Mobile robot positioning: Sensors and techniques. Journal of Robotic Systems, Special Issue on Mobile Robots 14(4), 231–249 (1997)

[2] Nistér, D., Naroditsky, O., Bergen, J.R.: Visual odometry. In: Conference on Computer Vision and Pattern Recognition (CVPR 2004), pp. 652–659 (2004)

[3] Olson, C.F., Matthies, L.H., Schoppers, M., Maimone, M.W.: Rover navigation using stereo ego-motion. Robotics and Autonomous Systems 43(4), 215–229 (2003)

[4] Makadia, A., Daniilidis, K.: Correspondenceless ego-motion estimation using an imu. In: IEEE International Conference on Robotics and Automation, pp. 3534–3539 (2005)

[5] Oskiper, T., Zhu, Z., Samarasekera, S., Kumar, R.: Visual odometry system using multiple stereo cameras and inertial measurement unit. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007), June 17-22, pp. 1–8 (2007)

[6] Kurazume, R., Nagata, S., Hirose, S.: Cooperative positioning with multiple robots. In: The IEEE International Conference in Robotics and Automation, pp. 1250–1257 (1994)

[7] Rekleitis, I.M., Dudek, G., Milios, E.E.: Multi-robot cooperative localization: a study of trade-offs between efficiency and accuracy. In: The IEEE/RSJ International Conference on Intelligent Robots and System, vol. 3, pp. 2690–2695 (2002)

[8] Mourikis, A.I., Roumeliotis, S.I.: Performance analysis of multirobot cooperative localization. IEEE Transactions on Robotics 22(4), 666–681 (2006)

[9] Roumeliotis, S.I., Bekey, G.A.: Distributed multirobot localization. IEEE Transactions on Robotics and Automation (5), 781–795 (2002)

[10] Mueller, S., Tsang, R.P., Ghosal, D.: Multipath routing in mobile ad hoc networks: Issues and challenges. In: Calzarossa, M.C., Gelenbe, E. (eds.) MASCOTS 2003. LNCS, vol. 2965, pp. 209–234. Springer, Heidelberg (2004)

[11] Spanos, D.P., Olfati-Saber, R., Murray, R.M.: Approximate distributed kalman filtering in sensor networks with quantifiable performance. In: 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005) (2005)

[12] Alriksson, P., Rantzer, A.: Distributed kalman filtering using weighted averaging. In: 17th International Symposium on Mathematical Theory of Networks and Systems (MTNS) (2006)

[13] Olfati-Saber, R.: Distributed kalman filtering for sensor networks. In: 46th IEEE Conference on Decision and Control (December 2007)

[14] Carli, R., Chiuso, A., Schenato, L., Zampieri, S.: Distributed kalman filtering using consensus strategies. In: 46th IEEE Conference on Decision and Control (December 2007)

[15] Alriksson, P., Rantzer, A.: Experimental evaluation of a distributed kalman filter algorithm. In: 46th IEEE Conference on Decision and Control (December 2007)

[16] Zhang, P., Martonosi, M.: LOCALE: collaborative localization estimation for sparse mobile sensor networks. In: International Conference on Information Processing in Sensor Networks (IPSN), pp. 195–206 (2008)

[17] Mendel, J.M.: Lessons in Estimation Theory for Signal Processing, Communications and Control. Prentice-Hall, Englewood Cliffs (1995)

[18] Barooah, P., da Silva, N.M., Hespanha, J.P.: Distributed optimal estimation from relative measurements for localization and time synchronization. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 266–281. Springer, Heidelberg (2006)

[19] Barooah, P., Hespanha, J.P.: Estimation from relative measurements: Algorithms and scaling laws. IEEE Control Systems Magazine 27(4), 57–74 (2007)
[20] Rhodes, I.: A tutorial introduction to estimation and filtering. IEEE Transactions on Automatic Control 16, 688–706 (1971)
[21] Barooah, P.: Estimation and Control with Relative Measurements: Algorithms and Scaling Laws. PhD thesis, University of California, Santa Barbara (July 2007)
[22] Anderson, B.D.O., Moore, J.B.: Optimal Filtering. Dover, Mineola (2005)
[23] Barooah, P., Hespanha, J.P.: Distributed optimal estimation from relative measurements. In: Proceedings of the 3rd International Conference on Intelligent Sensing and Information Processing (ICISIP), December 2005, pp. 226–231 (2005)
[24] Golub, G.H., van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)

Thermal-Aware Sensor Scheduling for Distributed Estimation

Domenic Forte and Ankur Srivastava

University of Maryland
Department of Electrical and Computer Engineering
College Park, Maryland
dforte@umd.edu, ankurs@umd.edu

Abstract. Recent work has shown that rising temperatures are increasing failures and reducing integrated circuit reliability. Although such results have prompted development of thermal management policies for stand-alone processors and on distributed power management, there is an overall lack of research on thermal management policies and their tradeoffs in sensor networks where sensors can overheat due to excessive sampling. Our primary focus in this paper is to examine the relationship between sampling, number of sensors, sensor node temperature, and state estimation error. We devise a scheduling algorithm which can achieve a desired real-time performance constraint while maintaining a thermal limit on temperature at all nodes in a network. Analytical results and experimentation are done for estimation with a Kalman filter for simplicity, but our main contributions should easily extend to any form of estimation with measurable error.

1 Introduction

1.1 Motivation

Rising temperatures are increasing failures and reducing integrated circuit reliability. In particular, [1] discusses thermal related effects such as electromigration, stress migration, and time-dependent dielectric breakdown and analyzes each effect's contribution to an IC's mean time to failure. Typically, such IC systems are given a manufacturer specified constraint to limit operating temperature and prevent permanent hardware damage. Traditionally, thermal issues within a chip have been handled at the package level with sophisticated but expensive cooling methods. However, current work is focusing on dynamic thermal management policies for processors and stand-alone IC systems that can adapt resource utilization and power dissipation to meet system demands. For example, dynamic voltage and frequency scaling is a technique whereby the voltage or frequency of a microprocessor can be automatically adjusted to conserve power or to reduce the amount of heat generated by the chip. This is at the cost of performance since the number of instructions that can be executed in a given amount of time is reduced as a result.

The majority of existing work addresses the thermal management problem in stand-alone systems. However, recent reports [2] emphasize the need to perform thermal management across stand-alone system boundaries, such as those in data centers. A sensor network represents another distributed platform where individual systems (i.e. sensor nodes) collaboratively solve complex tasks is an autonomous fashion. Unlike data-centers, the temperature between compute nodes in generally uncorrelated as sensor nodes are very far apart. However, sensor nodes are smaller and cheaper self-contained systems that cannot be cooled by fans or fluids due to cost- and energy-related resource constraints. Therefore, without explicit management of some kind, they may overheat under heavy workloads which includes obtaining measurements, processing or compressing those measurements, and communicating measurements and results. Note that most sensor network applications may not possess heavy enough workloads to require thermal management, but visual sensor networks (VSN) are one exception. VSN's require cameras (arrays of sensors) to acquire vast amounts of image data, run intensive computer vision algorithms to extract information or compress image data, and communicate results and/or images across bandwidth-limited channels.

Ignoring sensor node temperature could result in failure of any one the node's hardware components for sensing, processing, or communication which would require maintenance or replacement. When one considers sensor networks that operate in remote or restricted regions and may not be easily accessible, replacement is not a feasible solution. Aside from permanent damage, the measurements obtained by overheated sensing hardware and perhaps even calculations done by the processor could be corrupted by noise affecting the distributed computation of the entire sensor network.

Sensor networks have other unique characteristics as well. They are commonly formed via wireless communication, so power and temperature could easily be affected by dynamic channel conditions. The underlying structure of the network plays a key part in how energy is dissipated and consequently temperature behaves as well. For example, in a centralized wireless network with direct communication between nodes and some centralized hub, nodes farther away must dissipate considerably more energy than nodes closer to the center. Alternatively, in a multi-hop communication scenario where nodes communicate to the center through each other, the opposite is the case. Many sensor network applications in monitoring, tracking, and estimation also come with real-time constraints. Therefore, in sensor networks, temperature in nodes may not be coupled in the same sense as data-centers and multi-core processors, but is highly dependent on communication, desired global performance, and local scheduling. We feel that such complications make thermal management in sensor networks an interesting, challenging, and relevant research agenda and hope to explore some avante-garde solutions to them in this work and future research.

Existing work in sensor networks deals with resource constraints such as power or energy [3] to increase network lifetime, but not temperature. While a correlation between power and temperature exists, adhering to a power related requirement, such as battery life, can still result in a failure to meet thermal constraints. This is a consequence of one key difference between temperature and power. Unless there is some kind of energy harvesting mechanism, power in any system does not get replenished whereas temperature is a dynamically varying quantity and depends on the current power density. To clarify, consider a sensor network where nodes have varying levels of energy remaining. Most energy-aware management policies would re-distribute the workload among nodes so that those with the highest remaining energy are given greater workloads. While this certainly would extend the lifetime of the network, a constraint on temperature at the harder working nodes could easily be violated resulting in hardware damage. This would actually decrease the lifetime and reliability of the entire network even though the power constraints are satisfied. However, if the problem were re-cast in a thermal aware setting, the goal would be to schedule work at nodes over time such that no constraint is ever violated, but network lifetime is still maximized.

Our primary goal is to develop a thermal management policy to improve sensor reliability by constraining sensor temperature and examine its performance trade-offs. In this work, we focus on the performance of sensor networks that gather data individually, but collaboratively compute a state estimate. The primary focus is on the relationship between sampling, number of sensors, local sensor node temperature, and the global state estimation error. We devise a scheduling algorithm to achieve a desired real-time performance constraint while maintaining a thermal limit on temperature at all nodes in the network. For simplicity, we develop our ideas for a distributed Kalman filter (KF) architecture estimating a linear time-invariant system, but our policies should extend to any estimation technique where an error can be reliably estimated over time. The secondary goal of this work is to motivate future research to develop thermal management techniques for distributed systems.

1.2 Contributions and Assumptions

Throughout this work, we consider a network where sensor nodes obtain and directly transmit their measurements to a single KF hub estimating a linear time-invariant system. We assume that all nodes in the network have identical hardware and are measuring the same process. Therefore, the thermal characteristics, system dynamics, and measurement noise are considered identical for each node and independent among nodes. These assumptions are made to simplify the notation and calculations, but the underlying structure of the solutions should extend regardless.

Common solutions to thermal reliability issues include redundancy and graceful performance degradation. We provide a simple method to find the number of redundant sensors needed along with their sampling rate and temporal staggering to satisfy both thermal and performance related system constraints. Rather than exhaustively test every scheme, we offer a rationale for omitting certain configurations. Performance in our case refers to the confidence in state estimates. Therefore, in order to compare the performance of any two solutions, we obtain steady state estimation error for both. This is a solid basis for comparison provided that the system allows convergence to a stable state (see detectablility in [4]) and the filters run long enough to reach it. Results show that our policy can successfully balance the tradeoffs between thermal and performance related constraints.

The rest of the paper is organized as follows: Section 2 contains a brief overview of the Kalman Filter and a review of thermal modeling in electronic systems. Sections 3 and 4 gives an extended look at the KF's operation and sensor thermal behavior with sub-sampling and additional sensors considered. Section 5 summarizes the algorithms used to implement our policy. Section 6 contains a radar tracking example to evaluate our methods. We conclude with section 7.

2 PRELIMINARIES

2.1 The Kalman Filter (KF)

The KF is a set of mathematical equations that recursively estimate the state of a process, system, or environment by assuming the Markov property where the true state is conditionally independent of all earlier states given the prior state. Any KF operation begins with a system description of the process and measurement models:

$$\mathbf{s}_i = A\mathbf{s}_{i-1} + \mathbf{q}_{i-1} \tag{1}$$

$$\mathbf{z}_i = H\mathbf{s}_i + \mathbf{u}_i \tag{2}$$

where $\mathbf{s}_i$ denotes the system state and $\mathbf{z}_i$ corresponds to the sensor measurement for each time step i. $\mathbf{s}_i$ is calculated recursively using a linear combination of the previous state $\mathbf{s}_i$ and random system dynamics (sometimes referred to as process noise) $\mathbf{q}_{i-1}$. $\mathbf{z}_i$ is assumed to be related to the current system state by H and measurement noise $\mathbf{u}_i$. Note that $q_i \sim N(0, Q)$, $u_i \sim N(0, U)$, and we will assume that q_i and u_i are uncorrelated $\forall\, i$.

The KF is composed of two distinct phases at each discrete time-step i: predict and correct. The predict phase uses the state estimate from the previous time-step to produce an estimate of the state at the current time-step. In the correct phase, sensor information at the current time-step is used to refine this prediction for a more accurate state estimate. The two predict and three correct phase equations are:

$$\hat{\mathbf{s}}_{i|i-1} = A\hat{\mathbf{s}}_{i-1|i-1} \tag{3}$$

$$P_{i|i-1} = AP_{i-1|i-1}A^T + Q \tag{4}$$

$$\hat{\mathbf{s}}_{i|i} = \hat{\mathbf{s}}_{i|i-1} + K_i(\mathbf{z}_i - H\hat{\mathbf{s}}_{i|i-1}) \tag{5}$$

$$P_{i|i} = (I - K_iH)P_{i|i-1} \tag{6}$$

$$K_i = P_{i|i-1}H^T(HP_{i|i-1}H^T + U)^{-1} \tag{7}$$

$\hat{\mathbf{s}}_{i|i}$ and $\hat{\mathbf{s}}_{i|i-1}$ represent the KF's corrected and predicted state estimates given $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_i$ and given $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{i-1}$ respectively while $P_{i|i-1}$ and $P_{i|i}$ are the respective error covariances (i.e. uncertainty in the estimate). The Kalman gain matrix K_i controls the contributions of $\hat{\mathbf{s}}_{i|i-1}$ and $\mathbf{z}_i$ to $\hat{\mathbf{s}}_{i|i}$. The derivation of these general equations can be a found in any standard textbook such as [4]. Extensions are also available for a distributed architecture [5]. In this work, the type of distributed network under consideration is one where sensor nodes obtain and directly transmit their local measurements to a central KF node estimating the global system state.

In this paper, we are primarily concerned with the steady state covariance P_∞. In short, P_∞ is what the estimate uncertainty converges to as $i \to \infty$. When the error has converged, the Kalman gain essentially becomes a constant and optimally balances the contributions of the previous state and the current sensor measurements to the estimated state. P_∞ is important to us because it provides us with an upper bound on estimate uncertainty and allows us establish any changes in KF fidelity under different network parameters such as sensor sampling rate and number of sensor measurements.

2.2 Sensor Thermal Behavior

We define a sensor to be an electrical system made up of devices that measure a system state, communicate measurements to a central hub, and perform some light-weight processing. An example would be a processor with radio communication hardware. The thermal behavior of such electrical systems is often modeled by an RC circuit [6,7] with voltage representing the temperature and current representing power dissipation (see Figure 1(e)). The resistance R is the potential heat path throughout the package, while capacitance C indicates the ability of the processor to store heat [7].

The RC thermal model consists of the following relationship between the core temperature T and power consumption $p(t)$:

$$\frac{dT}{dt} = -\frac{T}{RC} + \frac{p(t)}{C} \tag{8}$$

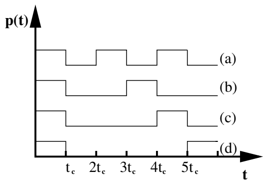
By running at a constant power $p_c \;\forall\; t$ and assuming an initial temperature $T(0) = T_{\text{init}}$, we solve Equation (8) and find that:

$$T(t) = T_{\text{nat}} + T_{\text{diff}} e^{-t/RC} \tag{9}$$

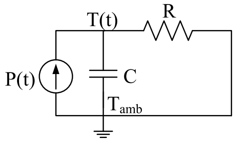
where $T_{\text{diff}} = T_{\text{init}} - T_{\text{nat}}$ and $T_{\text{nat}} = Rp_c$. Clearly, as $t \to \infty$, $T \to T_{\text{nat}}$.

3 Single Node Case

Consider the case with a single sensor that obtains and communicates its measurements to a centralized hub. Assume that the sensor dissipates $e_c = p_c \times t_c$ units of energy to accomplish this sensing/communication task. Then assume



(a-d)



(e)

Fig. 1. Power profiles for (a)$x = 2$, (b)$x = 3$, (c)$x = 4$, and (d)$x = 5$. (e) shows the RC thermal model.

that the sensor rests for $(x - 1)t_c$ additional units of time. The goal of this section is to determine if there exists an x to simultaneously meet a thermal constraint c_t limiting temperature and KF performance constraint c_p which establishes how much estimation error can be tolerated.

3.1 Thermal Analysis

A manufacturer typically provides a constraint on the temperature that if violated will jeopardize the electronic system's reliability. In the context of this paper, this means that $T(t)$ should not exceed c_t. To satisfy the condition on average, our thermal policy calls for a reduction in the sensor's sampling rate (i.e. increase in x). In this section, we will address the potential thermal savings.

The sensor sampling rate is reduced so that measurements are obtained and sent every x steps each of duration t_c, where x is an integer. To evaluate this thermal policy, let $p(t)$ be a modulated power signal for the sensor such that p_c is dissipated for t_c time units during the active (sampling) state followed by a shutdown state where power dissipated is zero (static power consumption ignored for simplicity) for $(x - 1)t_c$ additional time units. See Figure 1(a-d) for further clarification. In the first interval $0 \leq t \leq t_c$, the sensor dissipates p_c and T follows Equation (9). Then, the final temperature in the first interval $T(t_c)$ (determined by Equation (9)) is used as the initial temperature for the next interval $t_c < t \leq xt_c$. The solution of (8) under these conditions now gives:

$$T(t) = T(t_c)e^{-(t-t_c)/RC},\ t_c \leq t \leq xt_c \tag{10}$$

where $T(t_c) = T_{\text{nat}} + T_{\text{diff}}e^{-\frac{t_c}{RC}}$. We consider a periodic sampling where the interval pairs of active/shutdown repeat. During the i^{th} active/shutdown interval $(ixt_c < t \leq (i+1)xt_c)$, $T(t)$ actually obeys the same relationship as the first sampling interval $(0 < t \leq xt_c)$. Note that in this case, the final temperature of the active (shutdown) state acts as the initial temperature of the shutdown (active) state. Letting $T_{\text{diff}_i} = T_{\text{init}_i} - T_{\text{nat}}$ and $T_{\text{init}_{i+1}} = T(ixt_c)\ \forall\, i$ and generalizing Equations (9) and (10) in terms of i and x, we find that when a sensor is active $(p(t) = p_c)$

$$T(t) = T_{\text{nat}} + T_{\text{diff}_i}e^{-(t-(i-1)xt_c)/RC},\ ixt_c \leq t \leq (ix+1)t_c \tag{11}$$

and when the sensor is resting $(p(t) = 0)$.

$$T(t) = T_{\text{nat}} e^{-(t-((i-1)x+1)t_c)/RC} + T_{\text{diff}_i} e^{-t/RC},\ (ix+1)t_c \le t \le (i+1)xt_c \quad (12)$$

where

$$T_{\text{diff}_i} = T_{\text{nat}} \left(e^{-(t_c(x-1))/RC} - 1 \right) + T_{\text{diff}_{i-1}} e^{-xt_c/RC},\ i > 1 \quad (13)$$

For our purposes, the sensor is operating for a long time so we are most interested in the stable state behavior of these equations as $t \to \infty$ or equivalently as $i \to \infty$. In the stable state, the temperature within each pair of intervals will be the same. Therefore, we evaluate the average temperature as $i \to \infty$ by only including the behavior within one active/shutdown period:

$$\begin{aligned}
T_{\text{avg}_\infty}(x) &= \lim_{i\to\infty} \frac{1}{xt_c} \int_{(i-1)xt_c}^{ixt_c} T(t)dt = \frac{1}{xt_c} \int_0^{xt_c} T(t)|_{T_{\text{diff}_\infty}} dt & (14)\\
&= \frac{1}{xt_c} \left(\int_0^{t_c} T(t)dt + \int_{t_c}^{xt_c} T(t)dt \right) & (15)\\
&= \frac{1}{xt_c} \Big(T_{\text{nat}} t_c + RCT_{\text{diff}_\infty}(1 - e^{-t_c/RC}) + & (16)\\
&\quad RCT_{\text{nat}}(1 - e^{(x-1)t_c/RC} + RCT_{\text{diff}_\infty}(e^{-t_c/RC} - e^{-xt_c/RC}) \Big) & (17)
\end{aligned}$$

T_{diff_∞} is a critical parameter and by knowing its value we can calculate T_{avg_∞}. Fortunately, Equation (13) is a simple recurrence relation and it can be shown (see Section 7) that

$$T_{\text{diff}_\infty} = \lim_{i\to\infty} T_{\text{diff}_i} = T_{\text{nat}} \left(\frac{e^{-t_c(x-1)/RC} - 1}{1 - e^{-xt_c/RC}} \right) \quad (18)$$

Then, substituting Equation (18) into Equation (17) and canceling, we find that

$$T_{\text{avg}_\infty}(x) = \frac{T_{\text{nat}}}{x} \quad (19)$$

Note that $T_{\text{avg}_\infty} \to 0$ as $x \to \infty$ which means the temperature is approaching the environment's ambient temperature T_{amb}. This is accurate behavior when one considers a sensor that is active for t_c time units then rests forever. Figure 2(a) shows T_{avg_∞} offset by the ambient temperature (we have chosen room temperature 298 K) plotted against x.

In this paper, we will restrict a sensor's T_{avg_∞} by a thermal constraint. With any given c_t, we now define a sub-sampling rate by x^*:

$$x^* = \min x \text{ s.t. } T_{\text{avg}_\infty}(x) < c_t \quad (20)$$

Operating a sensor with $x \ge x^*$ guarantees the sensor's temperature remains below the thermal constraint on average.

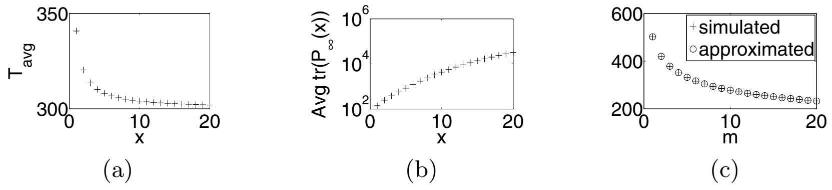


Fig. 2. (a) shows $T_{\text{avg}_{\infty}(x)}$ (in Kelvin) for typical R, C, and p_H values obtained from Hotspot [8]. (b) shows $\text{tr}(P_{\text{avg}_{\infty}}(x))$ on a log-linear scale and (c) compares $\text{tr}(P_{\infty}(m))$ and $\hat{P}(m)$. Both plots are from the radar tracking example in section 6.

3.2 Sub-sampled KF

In this section, we examine the tradeoffs in KF performance from sub-sampling. The goal is to find a sampling defined by x needed to fulfill a performance constraint. For the sake of illustration, we assume that $x = 2$ and obtain a version of the KF with sampling every 2 steps. The new process model including the current state $\mathbf{s}_i$ expressed as a function of the previous state $\mathbf{s}_{i-2}$ is:

$$\begin{aligned} \mathbf{s}_i &= A\mathbf{s}_{i-1} + \mathbf{q}_{i-1} && (21)\\ &= A\left(A\mathbf{s}_{i-2} + \mathbf{q}_{i-2}\right) + \mathbf{q}_{i-1} && (22)\\ &= A^2\mathbf{s}_{i-2} + A\mathbf{q}_{i-2} + \mathbf{q}_{i-1} && (23) \end{aligned}$$

This result can be generalized for any sub-sampling rate x:

$$\mathbf{s}_i(x) = A^x\mathbf{s}_{i-x} + \mathbf{q}_{i,x} \tag{24}$$

where $\mathbf{q}_{i,x} = \sum_{n=0}^{x-1} A^n\mathbf{q}_{i-n-1}$. Note that the measurement model remains unchanged, but measurements are only sensed/communicated every x steps.

Given the new process model, the modified predict and correct phase equations are:

$$\begin{aligned} \hat{\mathbf{s}}_{i|i-x} &= A^x\hat{\mathbf{s}}_{i-x|i-x} && (25)\\ P_{i|i-x} &= A^xP_{i-x|i-x}A^{x^T} + Q(x) && (26)\\ K_{i,x} &= P_{i|i-x}H^T(HP_{i|i-x}H^T + R)^{-1} && (27)\\ \hat{\mathbf{s}}_{i|i} &= \hat{\mathbf{s}}_{i|i-x} + K_{i,x}(\mathbf{z}_i - H\hat{\mathbf{s}}_{i|i-x}) && (28)\\ P_{i|i} &= (I - K_{i,x}H)P_{i|i-x} && (29) \end{aligned}$$

where $Q(x) = \sum_{k=0}^{x-1} A^kQA^{k^T}$ and was obtained by assuming $\text{cov}(\mathbf{q}_i, \mathbf{q}_j) = 0$ (system dynamics are uncorrelated) $\forall\ i \neq j$.

In this paper, we are primarily concerned with understanding the behavior of the steady state covariance P_{∞} and its relationship to sampling rate. Therefore, existence of the steady state is a requirement for applying these results. Finding an analytic expression for a multi-rate system can be difficult or even impossible for some cases. Therefore, we **simulate** our systems until $P_\infty(x)$ is reached. To make a fair comparison between the standard KF (sampling at every it_c) and the sub-sampled KF (sampling at every ixt_c), we need to include all sample points from the standard KF. Therefore, for the sub-sampled KF, we only perform the prediction step (rather than predict and correct) for the missing samples. Their error covariances correspond to predicted error covariances. Let us denote timesteps for missing samples by k where $1 \leq k \leq x-1$. We take an average trace (sum of the elements on the main diagonal) of the covariances and define:

$$P_{\text{avg}_\infty}(x) = \tfrac{1}{x}\,\text{tr}\left(P_\infty(x) + \sum_{k=1}^{x-1}(A^k P_\infty(x) {A^k}^T) + Q(x)\right) \tag{30}$$

which is shown in Figure 2(b) for the radar example in section 6. To maintain KF fidelity, we constrain the system with a performance constraint c_p. Then for any given c_p, we can define a sub-sampling rate x':

$$x' = \max x \text{ s.t. } P_{\text{avg}_\infty}(x) < c_p \tag{31}$$

If $x' \geq x^*$, any x chosen between x^* and x' results in a node operating below the thermal constraint while maintaining the required level of performance. If $x' < x^*$, sub-sampling alone will not be enough to fulfill the desired constraints. The next section will deal with this shortcoming.

4 Multiple Sensor Case

In this section, we study the benefits of using multiple sensors to estimate the same state and how this method may be used to solve our constrained problem. The goal is to find the number of sensors needed to fulfill a performance constraint along with their sampling rate and schedule to also meet the thermal constraint. Rather than exhaustively test every scheme, we offer a rationale for omitting certain configurations.

Let us first examine the potential improvements in error from using m identical sensors to estimate the same state. In section 3.2, the sub-sampled KF called for a modified process model. Here, the opposite is needed. Since sensors are identical, we will assume identical noise covariances U and that the H matrix relating current state $\mathbf{s}_i$ to the k^{th} sensor's measurement $\mathbf{z}_{i,k}$ are the same as well. We will also assume $\text{cov}(\mathbf{u}_{i,j}, \mathbf{u}_{i,k})\ \forall\ j \neq k$ (uncorrelated noise between sensors). The modified measurement model is given by:

$$\mathbf{z}_i = H_{mult}(m)\mathbf{s}_i + \mathbf{u}_i \tag{32}$$

where $\mathbf{z}_i = \left(\mathbf{z}_{i,1} \dots \mathbf{z}_{i,m}\right)^T$, $H_{mult}(m) = \left(H \dots H\right)^T$, $u_i \sim N(\mathbf{0}, U_{mult}(m))$, and $U_{mult}(m) = \begin{pmatrix} U & 0 & 0 \\ & \ddots & \\ 0 & \ddots & 0 \\ 0 & 0 & U \end{pmatrix}$. Note that $H_{mult}(m)$ and $U_{mult}(m)$ contain m total

H and U block matrices respectively. For a system with synchronous sensors operating in a centralized manner, its performance is the same as a system with one composite sensor with smaller variance [9]. This means that under these assumptions, $P_{i|i-1}$ and $P_{i|i}$ can be obtained by using Equations (4) and (6) with U replaced by $\frac{1}{m}U$. Note that this approximation will be useful in the results section to simplify simulations.

4.1 Thermal Analysis

Throughout this section, consider a multi-sensor network containing m identical sensors that are sufficiently distant from one another. Then it is sensible to assume that each sensor is thermally independent. Also, assume c_t and T_{amb} will be the same for identical sensors so the x^* calculated previously is sufficient for condition $T_{\text{avg}_\infty}(x) < c_t$ to hold for every sensor in the network. Our analysis can be extended to support varying c_t and T_{amb} in future works.

4.2 Multi-sensor Covariance

It is possible to obtain P_∞ for any m through simulation or closed form solution if it exists, but this becomes less computationally efficient as the system global state or measurement vectors grow. Instead we provide an analytical method to approximate $\text{tr}(P_\infty)$ by $\hat{P}$ for any m which seems to accommodate general system solutions rather well. We have observed that for our assumptions, the covariance seems to follow a decay proportional to $1/m$. Therefore, the approximation is given by:

$$\hat{P}(m) = \frac{\text{tr}(P_\infty(1))}{m^k} + \lim_{m\to\infty} \text{tr}(P_\infty(m)) \tag{33}$$

The first term in the above accounts for the decay and the second term acts as an offset which accounts for the error always introduced by process noise covariance Q. k is a positive constant which depends strictly on problem parameters. We find that one way to approximate k is to simulate the KF for several different choices of m to obtain $\text{tr}(P_\infty(m))$, solve for k after each simulation, and take an average. Figure 2(c) gives a plot of $\text{tr}(P_\infty(m))$ and $\hat{P}(m)$ on the same set of axes for the radar experiment in Section 6 to validate the approximation to the reader.

4.3 Policy Synthesis

We are interested in a method for finding the total number of sensors and the percentage of sensors that need to be active at any given time to fulfill a given c_t and desired c_p. We assume a policy that calls for the same number of active sensors at every time-step because uniform staggering of identical sensors was proven in [9] to be optimal for radar tracking. Intuitively, the consistency in the state estimate through uniform sampling should at least be close to the best possible estimate even for more general problems. In fact, [9] also provided numerical results for systems consisting of 2 sensors with different measurement noise variance and found that uniform staggering is no more than 30% worse than the optimal scheme even with a noise ratio among sensors as large as 10.

Let m be the number of active sensors. Then by comparing $\hat{P}(m)$ with c_p, we will obtain the number of sensors, m', needed at any given step to maintain KF performance:

$$m' = \min m \text{ s.t. } \hat{P}(m) < c_p \tag{34}$$

c_t enforces that any sensor only be active every x^* steps so the total number of sensors required in this approach is $x^* m'$.

5 Design Methodology

5.1 Single Sensor Algorithm

Our first algorithm restricts itself to the steady state operation of single sensor and returns the sensor's sampling rate x_s. While the thermal constraint is met, this method requires applications that allow for relaxed performance constraints. The steps are:

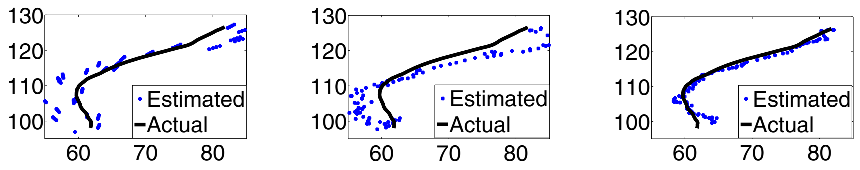
1. Compute $T_{\text{nat}} = Rp_c$
2. Solve Equation (20) to obtain x^*.
3. Let $x_{\text{test}} = x^*$.
4. Simulate the system with the KF system parameters A, H, Q, U and obtain $P_{\text{avg}_\infty}(x_{\text{test}})$ from (30).
5. Solve Equation (31) for x'.
6. If $x' \neq x_{\text{test}} - 1$, increment x_{test} and go back to step 4.
7. If $x' \geq x^*$, return $x_s = x'$. Otherwise, return 0 because the performance constraint cannot be met.

Upon failure of this algorithm, one may relax the performance constraint (if possible) or apply our second algorithm.

5.2 Multiple Sensor Algorithm

The second algorithm considers additional sensors so that performance is maintained while meeting thermal constraints. This method returns the total number of sensors needed m_s and their sampling rates x_s. The steps are:

1. Compute $T_{\text{nat}} = Rp_c$
2. Evaluate Equation (33) with a suitable k to obtain $\hat{P}(m)$.
3. Solve Equation (34) to obtain m'.
4. Solve Equation (20) to obtain x^*.
5. Return $m_s = x^* m'$ and $x_s = x^*$.



(a)$x = 4$, $m = 1$ (b)$x = 1$, $m = 1$ (c) $x = 1$, $m = 16$

Fig. 3. (a,b,c) Estimated trajectories plotted against true trajectory for 3 (x, m) pairs

6 Simulation Results

Let us examine a 2D radar tracking example with random-walk velocity [10]. Imagine a radar fan beam rotating continually through 360° with a period T. The state vector $\mathbf{s}_i = (pos_{\mathrm{h},i},\ vel_{\mathrm{h},i},\ pos_{\mathrm{v},i},\ vel_{\mathrm{v},i})^T$ where $pos_{\mathrm{h},i}$, $pos_{\mathrm{v},i}$, $vel_{\mathrm{h},i}$, $vel_{\mathrm{v},i}$ are the target's horizontal and vertical positions and velocities at time step i. For the target's measured position, $\mathbf{z}_i = (pos_{\mathrm{h},i},\ pos_{\mathrm{v},i})^T$. For sampling rate x and one sensor, the generalized process and measurement models are given by:

$$\mathbf{s}_i = A^x \mathbf{s}_{i-x} + q_{i,x} \tag{35}$$

$$\mathbf{z}_i = H\mathbf{s}_i + \mathbf{u}_i \tag{36}$$

where $A^x = \begin{pmatrix} 1 & xT & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & xT \\ 0 & 0 & 0 & 1 \end{pmatrix}$ and $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$. This can easily be extended to a multiple sensor scenario by applying the methods described in Section 4. The state transition matrix A^x acts on $\mathbf{s}_{i-x}$ predicting $\mathbf{s}_i$ (which is x steps away) assuming constant velocity in either direction. For example, $pos_{\mathrm{h},i} = pos_{\mathrm{h},i-x} + xTvel_{\mathrm{h},i-x} + q_{i,x}$. It is the system dynamics or process noise $\mathbf{q}_{i,x}$ which accounts for randomness in the target trajectory (i.e. target accelerating or turning at any given time). In the single and multi-sensor cases, increasing x should clearly yield greater uncertainty in $\hat{\mathbf{s}}_{i|i-x}$. In the multi-sensor case, an increase in m should raise the confidence in $\hat{\mathbf{z}}_i$ and result in an improved estimate. Figures 3 (a-c) support these statements by showing the estimated trajectories for several cases of x and m.

In figure 4, we report algorithm results for the radar tracking example with varying thermal and performance constraints and fixed process and measurement noise covariances. Note that the opposite case with fixed constraints and varying noise covariances will highlight similar trends so the results are omitted. For the single sensor case, relaxing c_p increases x_s (lowering the sampling rate) and relaxing c_t decreases x_s (raising the sampling rate). These are an accurate trends for the opposing constraints. For the multi-sensor case, the required number of sensors, m_s, decreases with more relaxed c_t. It can be seen that our algorithms provide an m_s and x_s that balance both thermal and performance related constraints.

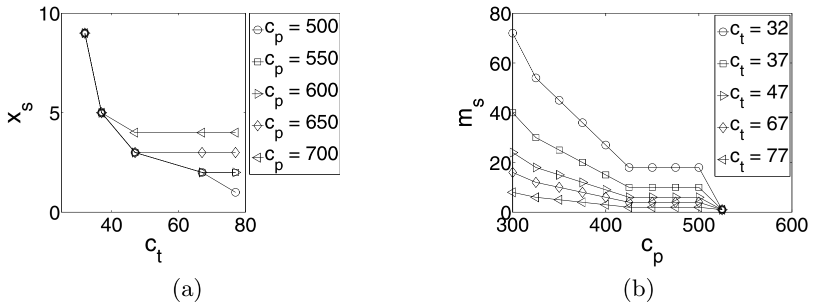


Fig. 4. (a) Single Sensor Case and (b) Multiple Sensor Case. Note: c_t is in ^{o}C in this figure.

7 Conclusion

The effects of a thermal management policy on the steady state Kalman filter were studied in this paper. The average temperature of sensors operating at lower sampling rate x^* was derived and compared with a manufacturer specified constraint to satisfy a thermal requirement. Then we met both thermal and desired performance related constraints by reducing the KF sampling rate to x_s or including m_s sensors to estimate the same process. Rather than simulate every filter for m sensors, we also contributed a general analytical method to approximate $\mathrm{tr}(P_\infty(m))$. Instead of evaluating every possible schedule, intuition gave us a reasonable uniform sensor staggering method to fulfill performance constraints. The analytical results and design methodology should extend to any form of estimation with measurable error.

References

1. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: The impact of technology scaling on lifetime reliability. In: DSN, pp. 177–186. IEEE Computer Society, Los Alamitos (2004)
2. Ramos, L., Bianchini, R.: C-oracle: Predictive thermal management for data centers. In: IEEE 14th International Symposium on High Performance Computer Architecture, HPCA 2008, February 2008, pp. 111–122 (2008)
3. Soro, S., Heinzelman, W.: A Survey of Visual Sensor Networks. Advances in Multimedia 21 (2009)
4. Simon, D.: Optimal State Estimation: Kalman, H Infinity and Nonlinear Approaches. Wiley & Sons, Chichester (2006)
5. Rao, B.S.Y., Durrant-Whyte, H.F., Sheen, J.A.: A fully decentralized multi-sensor system for tracking and surveillance. Int. J. Rob. Res. 12(1), 20–44 (1993)
6. Rao, R., Vrudhula, S., Chakrabarti, C.: An optimal analytical solution for processor speed control with thermal constraints. In: Proc. Intl. Symp. Low Power Electronics and Design (ISLPED), pp. 292–297 (2006)
7. Cohen, A., Finkelstein, L., Mendelson, A., Ronen, R., Rudoy, D.: On estimating optimal performance of cpu dynamic thermal management. IEEE Computer Architecture Letters 2(1) (2003)

8. Skadron, K., Stan, M.R., Sankaranarayanan, K., Huang, W., Velusamy, S., Tarjan, D.: Temperature-aware microarchitecture: Modeling and implementation. ACM Trans. Archit. Code Optim. 1(1), 94–125 (2004)
9. Niu, R., Varshney, P.K., Mehrotra, K., Mohan, C.: Temporally staggered sensors in multi-sensor target tracking systems. IEEE Transactions on Aerospace and Electronic Systems 41(3), 794–808 (2005)
10. Brookner, E.: Tracking and Kalman Filtering Made Easy. Wiley-Interscience, Hoboken (April 1998)

Appendix

Equation 13 can be shown by induction:

$$\begin{aligned}
T_{\text{diff}_1} &= T_{\text{init}_1} - T_{\text{nat}} \\
T_{\text{diff}_2} &= T_{\text{init}_2} - T_{\text{nat}} = T(xt_c) - T_{\text{nat}} \\
&= T_{\text{nat}} \left(e^{-(t_c(x-1))/RC} - 1 \right) + T_{\text{diff}_1} e^{-xt_c/RC} \\
&\vdots \\
T_{\text{diff}_i} &= T_{\text{nat}} \left(e^{-(t_c(x-1))/RC} - 1 \right) + T_{\text{diff}_{i-1}} e^{-xt_c/RC} \ \forall \, i > 1
\end{aligned}$$

Then to prove the recurrence, let us simplify the notation of Equation 13 with

$$\begin{aligned}
a_i &= T_{\text{diff}_i} \\
c &= T_{\text{nat}} \left(e^{-(t_c(x-1))/RC} - 1 \right) \\
d &= e^{-xt_c/RC}
\end{aligned}$$

making the recurrence relation

$$a_i = c + da_{i-1}$$

Note that c and d are simply constants. Next, we rewrite the recurrence relation in terms of a_1

$$a_i = c \sum_{j=0}^{i-2} d^j + a_1 d^{i-1}$$

As $i \to \infty$, the first term

$$\begin{aligned}
c \lim_{i \to \infty} \sum_{j=0}^{i-2} d^j &= c \frac{d^{i-1} - 1}{d - 1} \\
&= \frac{c}{1-d}, \ |d| < 1
\end{aligned}$$

and the second term clearly $\to 0$ for $|d| < 1$. Therefore,

$$\lim_{i \to \infty} a_i = \frac{c}{1-d} \to \lim_{i \to \infty} T_{\text{diff}_i} = T_{\text{nat}} \left(\frac{e^{-t_c(x-1)/RC} - 1}{1 - e^{-xt_c/RC}} \right)$$

Decentralized Subspace Tracking via Gossiping

Lin Li[1], Xiao Li[1], Anna Scaglione[1], and Jonathan H. Manton[2]

[1] University of California, Davis, USA
{llli,eceli,ascaglione}@ucdavis.edu
[2] University of Melbourne, Victoria, Australia
jmanton@unimelb.edu.au

Abstract. We consider a fully decentralized model for adaptively tracking the signal's principal subspace, which arises in multi-sensor array detection and estimation problems. Our objective is to equip the network of dispersed sensors with a primitive for online spectrum sensing, which does not require a central fusion node. In this model, each node updates its local subspace estimate with its received data and a weighted average of the neighbors' data. The quality of the estimate is measured by the total subspace mismatch of the individual subspace component estimates, which converge asymptotically in the Lyapunov sense.

1 Introduction

Subspace-based signal processing methods have been applied successfully to both the spatial and temporal spectral analysis. Early approaches are based on the batch singular value decomposition (SVD) or the eigenvalue decomposition (EVD) of the data matrices [2]. However, they are unsuitable for the adaptive signal processing because repeated SVD/EVD is required and, thus, they are very computationally expensive. For faster processing time, many adaptive subspace estimation techniques have been proposed, such as modified SVD/ED [6,12,13], Rank-one updating algorithm [14] and subspace estimation as a constrained or unconstrained optimization problem [9,10,11]. The aforementioned literatures have studied the subspace estimation problem in a centralized setting, where all the information is available at a single location. There are situations, however in a large sensor network, where the available information is not available centrally.

In these scenarios, communications place a premium on the cost of the computation, not only because of the raw data transmission cost, but also due to the need of maintaining reliable routes from the sensors to the fusion site, responding to failures and coordinating the data gathering process. Recently, several authors [3,8] have considered alternative avenues for computation by means of network diffusion using only local communications. These mechanisms require minimal, if any, network knowledge, and are resilient to link failures or changes in network topology. Examples of in-network signal processing via gossiping are [1,15,16,17,18,19]. In particular, new frameworks to study distributed stochastic optimization [20,22], Kalman filtering[21] and adaptive filtering [7,23] have emerged, providing an alternative to the centralized data fusion model.

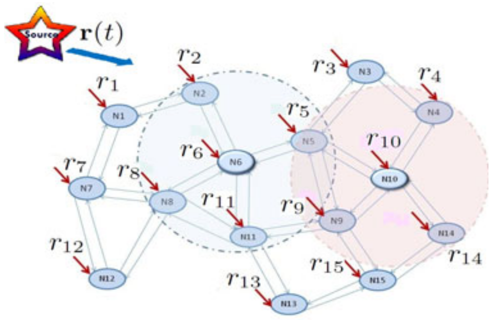


Fig. 1. Near-neighbor communication topology

Motivated by the significance of subspace estimation and tracking in signal processing, we focus our attention on the principal subspace tracking problem in a decentralized Wireless Sensor Network (WSN) setting. A typical WSN consists of a large number of sensors located in the environment being monitored. Each sensor has limited communication and computation resources. To overcome these limitations, a decentralized data fusion protocol is an appealing option given its resilience to node failure, mobility and its intrinsically low control overhead.

Fig.1 shows one possible information flow scheme of a decentralized WSN. To simplify the model, we assume that all the sensors in the network are identical and restricted to the near-neighbor communication with the same commnucation range. The network can be considered a random geometric graph, with radius equal to its communication range [24]. In this paper, we consider local communications performed through a static network and with synchronous updates. Randomized gossip on time-varying topologies are a natural generalization that will be pursued in a future study.

The organization of this paper is as follows. After briefly presenting the system's mathematical model and the average consensus protocol in the next subsections, we derive adaptive algorithms for tracking the signal's 1-dimensional and p-dimensional principal subspaces via gossiping, and provide stability analysis along with the numerical results in Section 2. To motivate our study, Section 3 presents a case study on spectrum sensing in a decentralized cognitive radio system. The application scenario considered for the proposed subspace tracking algorithm is to distributedly estimate the signal's subspace in a given radio frequency, and furthermore, to perform a decentralized detection on the spectrum's band occupancy. In the rest of the paper, the signal's principal subspace is referred as the subspace.

1.1 Problem Setup

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ represent an undirected graph associated to the given connected decentralized network. The set $\mathcal{N} = \{1, 2, \cdots, N\}$ denotes the nodes, and the set $\mathcal{E}$ is a collection of edges $\{i, j\}$ if the node i and j are connected for $i \neq j$. Denote $r_{i,t}$ as the i^{th} node's received data at time t and $\boldsymbol{r}_t = [r_{1,t}, r_{2,t}, \cdots, r_{N,t}]^T \in \mathbb{C}^N$

as the sample vector from the N sensors. The subspace tracking problem consists of adaptively computing the p-dimensional principal components of the data covariance matrix $\boldsymbol{R} = \mathbb{E}\{\boldsymbol{r}(t)\boldsymbol{r}^H(t)\} \in \mathbb{C}^{N\times N}$. The well-known Karhunen's method [6] for computing the dominating principal component $\boldsymbol{u}$, combined with Oja's learning rule [5] in a centralized network can be expressed as

$$\boldsymbol{u}_t = \frac{\boldsymbol{u}_{t-1} + \gamma_t \boldsymbol{r}_t \boldsymbol{r}_t^H \boldsymbol{u}_{t-1}}{\|\boldsymbol{u}_{t-1} + \gamma_t \boldsymbol{r}_t \boldsymbol{r}_t^H \boldsymbol{u}_{t-1}\|} \approx \boldsymbol{u}_{t-1} + \gamma_t \left[\boldsymbol{r}_t \boldsymbol{r}_t^H \boldsymbol{u}_{t-1} - \boldsymbol{u}_{t-1}\boldsymbol{u}_{t-1}^H \boldsymbol{r}_t \boldsymbol{r}_t^H \boldsymbol{u}_{t-1}\right] \quad (1)$$

where $\boldsymbol{u}_0$ is initialized to an arbitrary unit vector.

The computation of the learning rule in (1) requires access to the entire vector of observations $\boldsymbol{r}_t$. However, in a decentralized network, each node only knows its received data $r_{i,t}$. It is not possible for all the nodes to obtain the complete received signal vector, nor to carry out the computation in (1). Therefore, it is essential to seek a decentralized method to perform the computation with only the data that are available. Our objective is to provide a method to implement this learning rule in a decentralized fashion, by having each node compute via near-neighbor communications only the corresponding i^{th} component of the vector $\boldsymbol{u}_t$. The key idea is to decentralize the computation of the term $\boldsymbol{r}_t^H \boldsymbol{u}_{t-1}$ using as a primitive the so called class of average consensus gossiping protocols, which are going to be described in the next section.

1.2 Average Consensus Protocol

Recall that $\boldsymbol{r}^H\boldsymbol{u} = \sum_{i=1}^N r_i^* u_i$. Suppose $z_i(0) = r_i^* u_i$ is the known information at node i. The average consensus protocol [3,8] iteratively computes the average $\sum_{i=1}^N z_i(0)/N$ by performing a weighted average of the neighbors' data as follows

$$\boldsymbol{z}(k) = \boldsymbol{W}\boldsymbol{z}(k-1) \ , \quad (2)$$

given the initial condition $\boldsymbol{z}(0) = [z_1(0), z_2(0), \cdots, z_N(0)]^T$. The $(i,j)^{\text{th}}$ entry $\boldsymbol{W}_{ij}$ of the matrix $\boldsymbol{W}$ is the weight associated to the edge $\{i,j\}$, which is non-zero if and only if $\{i,j\} \in \mathcal{E}$, meaning node i and j are connected neighbors.

The undirected gragh $\mathcal{G}$ assumption implies $\boldsymbol{W}_{ij} = \boldsymbol{W}_{ji}$ and $\boldsymbol{W}_{ii} = 1 - \sum_{\{i,j\}\in\mathcal{E}} \boldsymbol{W}_{ij}$. Therefore, the matrix $\boldsymbol{W}$ has the same sparsity as the network graph, and it is a doubly stochastic matrix (*i.e.*, $\mathbf{1}_{N\times 1} = [1, 1, \cdots, 1]^T$ is a left and right eigenvector of $\boldsymbol{W}$). To achieve asymptotic average consensus, matrix $\boldsymbol{W}$ must also satisfy $\lim_{k\to\infty} \boldsymbol{W}^k = \frac{1}{N}\mathbf{1}\mathbf{1}^T = \boldsymbol{J}$ [3] or equivalently, the spectral radius satisfies $\rho(\boldsymbol{W} - \boldsymbol{J}) < 1$. This indicates the following,

$$\boldsymbol{z}(k) = \boldsymbol{W}^k \boldsymbol{z}(0) \xrightarrow{k\to\infty} \frac{1}{N}\sum_{i=1}^N z_i(0) \cdot \mathbf{1}_{N\times 1} \ . \quad (3)$$

The value $z_i(k)$ at each node approaches the averaged value of $\boldsymbol{z}(0)$ as k increases.

In general, in order for the functions of interest to be computed through the average consensus algorithm, they need to be separable in the form of $g(z) = g\left(\sum_{i=1}^N f_i(x_i)\right)$ and the value $\sum_{i=1}^N f_i(x_i)$ can be simply obtained by all the nodes via the average consensus algorithm. Setting the initial condition as $\boldsymbol{z}(0) = [f_1(x_1), \cdots, f_N(x_N)]^T$, then after the k^{th} iteration, we get

$$N\boldsymbol{z}(k) = N\boldsymbol{W}^k \left[f_1(x_1) \cdots f_N(x_N)\right]^T \xrightarrow{k\to\infty} \sum_{i=1}^{N} f_i(x_i)\mathbf{1}_{N\times 1} \ . \tag{4}$$

Remark 1. One possible construction of the weight matrix $\boldsymbol{W}$ is in terms of the Lapacian matrix $\boldsymbol{L}$, where $\boldsymbol{W} = \boldsymbol{I} - \delta\boldsymbol{L}$. The value, δ, is selected such that each element in $\boldsymbol{W}$ is non-negative.

2 Decentralized Subspace Tracking Algorithms

In this section, we propose the decentralized 1-dimensional and p-dimensional subspace tracking algorithms. The 1-dimensional tracking algorithm is based on the Karhunen-Oja's updating rule [5,6] as presented in (1). The p-dimensional tracking algorithm is based on the NOja's algorithm [9,10], which is described in Section 2.2. For each case, the algorithm is first explained and followed by the stability analysis via its associated ordinary differential equation (ODE). Furthermore, it can be shown that the equilibrium point of the ODE is stable under certain conditions and it spans the principal subspace of the signal's covariance matrix, as corroborated by the numerical results presented in each case.

2.1 Decentralized 1-Dimensional Subspace Tracking

Consider the problem of 1-dimensional subspace tracking based on Karhunen-Oja's learning rule in (1), but restricted to the near-neighbor communications. We propose decentralizing the learning rule using the average consensus protocol. Specifically, the nework first seeks to reach a *consensus* on the inner product $\boldsymbol{r}^H\boldsymbol{u}$ via gossiping, and then *updates* its principal component estimation using (1).

Let $x_i{=}r_{i,t}$ and $f_i(x_i){=}r^*_{i,t}u_{i,t}$, then $(f_1(x_1), \ldots, f_N(x_N))^T{=}\boldsymbol{r}^*_t{\circ}\boldsymbol{u}_t$, where $A{\circ}B$ is the element by element product. Assume each node performs k iterations of average consensus, from (4) it is clear that $\{N\boldsymbol{W}^k\left(\boldsymbol{r}^*_t \circ \boldsymbol{u}_t\right)\}$ are the estimates of the inner product $\boldsymbol{r}^H_t\boldsymbol{u}_t$ made at the individual nodes. Let $\boldsymbol{u}_{t,k}$ denote the subspace estimate at time t assuming k average consensus iterations are performed. Using the property, $\boldsymbol{W}(\boldsymbol{r}^*\circ\boldsymbol{u})\circ\boldsymbol{r} = (\boldsymbol{W}\circ\boldsymbol{r}\boldsymbol{r}^H)\boldsymbol{u}$, and the approximation,$\{N\boldsymbol{W}^k\left(\boldsymbol{r}^*_t \circ \boldsymbol{u}_t\right)\}$, in place of the actual $\boldsymbol{r}^H_t\boldsymbol{u}_t$, the decentralized Karhunen-Oja's learning rule can be reformulated into the following network wide update

$$\boldsymbol{u}_{t,k} = \boldsymbol{u}_{t-1,k} + \gamma_t\boldsymbol{d}_{t,k} \ , \tag{5}$$

$$\text{where} \ \ \boldsymbol{d}_{t,k} = N\boldsymbol{W}^k \circ \left[\left(\boldsymbol{I} - (N\boldsymbol{W}^k \circ \boldsymbol{u}_{t-1,k}\boldsymbol{u}^H_{t-1,k})\right)\boldsymbol{r}_t\boldsymbol{r}^H_t\right]\boldsymbol{u}_{t-1,k} \ .$$

Algorithm 1 summarizes the proposed 1-dimensional subspace tracking of $\boldsymbol{R}$. Here, $\boldsymbol{u}_{0,0}$ is initialized arbitrarily with the property that $\boldsymbol{u}^H_{0,0}\boldsymbol{u}_{0,0} \approx 1$. For convenience, we denote by $\text{AC}\left(f_i(x_i)\right)$ the i^{th} sensor's output of the average consensus algorithm given the initial value $f_i(x_i)$, and γ_t is the stepsize.

Algorithm 1. 1-Dimensional Subspace Tracking (at the i^{th} node)

```
input  : A sequence of $r_{i,t}$
output: The $i^{\text{th}}$ element of the principal component $\boldsymbol{u}$
1 Initialization: $t = 0$;
2   $u_i = \sqrt{1/N}$;
3 Recursion:
4   foreach $t = 1 : T$ do
5       $\text{ac}_i \leftarrow N \cdot \text{AC}(u_i r^*_{i,t})$;
6       $u_i \leftarrow u_i + \gamma_t(\text{ac}_i)(r_{i,t} - \text{ac}^*_i \cdot u_i)$;
7   end
```

Stability Analysis. The stability analysis of a stochastic approximation is closely related to the asymptotic property of the solutions to its associated ODE [27]. The stabilities of interest are the ODEs of $\boldsymbol{u}$ and $\boldsymbol{u}^H\boldsymbol{u}$. The former is to study the convergence of the subspace estimate and the latter is to test the unit norm property of the estimate. Under certain conditions, the subspace estimate $\boldsymbol{u}$ is globally asymptotically stable in the Lyapunov sense.

The associated ODE of the stochastic approximation of $\boldsymbol{u}$ derived in ([5]) is $d\boldsymbol{u}/dt = \mathbb{E}\{\boldsymbol{d}_{t,k}\}$. Assuming $\boldsymbol{r}_t$ is stationary, we can write $\mathbb{E}\{\boldsymbol{r}\boldsymbol{r}^H\} = \boldsymbol{R}$, and by construction $\boldsymbol{W}^k = \boldsymbol{J} + (\boldsymbol{W} - \boldsymbol{J})^k$. The ODE of $\boldsymbol{u}$ can be expressed as

$$\frac{d\boldsymbol{u}}{dt} = \mathbb{E}\{\boldsymbol{d}_{t,k}\} = (\boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^H)\boldsymbol{R}\boldsymbol{u} + \boldsymbol{\Delta}_{\text{ODE}}\boldsymbol{u} \ , \tag{6}$$

where, by setting $\tilde{\boldsymbol{W}} = \boldsymbol{W} - \boldsymbol{J}$,

$$\boldsymbol{\Delta}_{\text{ODE}} = N\left(\tilde{\boldsymbol{W}}^k \circ \mathbf{R} - \tilde{\boldsymbol{W}}^k \circ \boldsymbol{u}\boldsymbol{u}^H\boldsymbol{R} - (\tilde{\boldsymbol{W}}^k \circ \boldsymbol{u}\boldsymbol{u}^H)\boldsymbol{R}\right) + \mathcal{O}(N^2\tilde{\boldsymbol{W}}^{2k}) \tag{7}$$

represents the deviation of the decentralized ODE from the centralized ODE of $\boldsymbol{u}$. To show the subspace approximation $\boldsymbol{u}$ converges to the principal component of $\boldsymbol{R}$, we need to first prove the following Lemma.

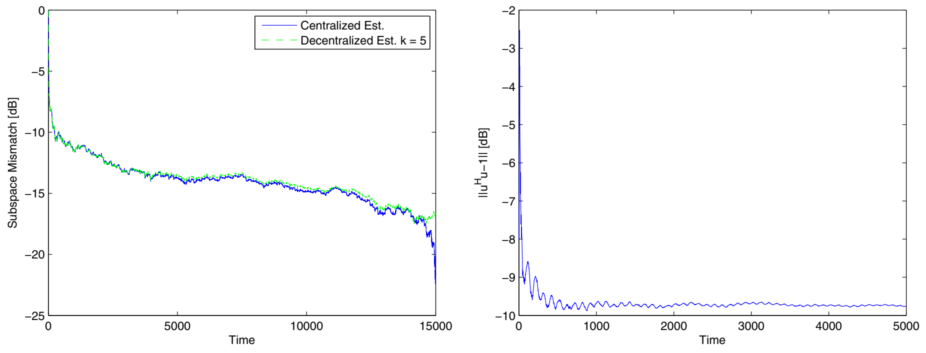
Lemma 1 (Stability of $\boldsymbol{u}^H\boldsymbol{u}$). *Let $\lambda_{2,w}$ be the 2nd largest eigenvalue of $\boldsymbol{W}$ and $\lambda_{2,w} < 1$ by construction. The ODE of $\boldsymbol{u}^H\boldsymbol{u}$ asymptotically converges to $1 + \mathcal{O}(N\lambda^k_{2,w})$ if the number of the average consensus iterations k is sufficiently large.*

Proof. See Appendix A. □

Notice that $\tilde{\boldsymbol{W}}^k \approx \mathbf{0}_{N\times N}$ for k sufficiently large. Hence, the ODE of $\boldsymbol{u}$ reduces to Oja's ODE, $d\boldsymbol{u}/dt = (\boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^H)\boldsymbol{R}\boldsymbol{u}$. Oja in [5] proves that $\boldsymbol{u}$ converges if $\boldsymbol{u}^H\boldsymbol{u} = 1$. Following similar arguments, Lemma 2 extends this result.

Lemma 2 (Convergence). *The 1-dimensional subspace estimate $\boldsymbol{u}$ converges to the principal component of the covariance matrix $\boldsymbol{R}$ asymptotically if $\boldsymbol{u}^H\boldsymbol{u} = 1 + \mathcal{O}(N\lambda^k_{2,w})$ for k sufficiently large.*

Proof. See Appendix B. □



(a) Convergence (b) Stability

Fig. 2. Numerical results for the 1-dimensional subspace tracking algorithm

Numerical Results. There are 10 sensors in the network. The source contains a *white noise* corrupted narrowband signal at 20dB SNR. Fig. 2a illustrates the performance of the decentralized 1-D subspace tracking algorithm. The average consensus protocol is performed at $k = 5$ iterations per update. The quality of the estimate is measured by the subspace mismatch, $\|(\boldsymbol{u}_{\text{true}}^{\perp})^H \boldsymbol{u}_{\text{est}}\|^2$, which is the Euclidean norm of the estimated 1-D subspace projected onto the non-dominating eigenvectors of $\boldsymbol{R}$, and it is compared to the *null* value. Fig. 2b shows the stability of this algorithm (Lemma 1), which is measured by $||\boldsymbol{u}^H\boldsymbol{u} - 1||$. We observe that the trajectory of $\boldsymbol{u}^H\boldsymbol{u}$ approaches 1 asymptotically.

2.2 Decentralized p-Dimensional Subspace Tracking

Oja's learning rule can be extended to a more general problem of tracking the p-dimensional subspace [5,9,10] of the signal's covariance. The derivation of the decentralized algorithm is based on the NOja's update [10] as shown below

$$\boldsymbol{U}_t = \boldsymbol{U}_{t-1} + \gamma_t \cdot \boldsymbol{D}_{t,k} \ , \tag{8}$$
$$\text{where} \ \ \boldsymbol{D}_{t,k} = \boldsymbol{r}_t\boldsymbol{r}_t^H\boldsymbol{U}_{t-1}(\boldsymbol{I} - \boldsymbol{U}_{t-1}^H\boldsymbol{U}_{t-1}) + (\boldsymbol{I} - \boldsymbol{U}_{t-1}\boldsymbol{U}_{t-1}^H)\boldsymbol{r}_t\boldsymbol{r}_t^H\boldsymbol{U}_{t-1} \ .$$

The subspace estimate $\boldsymbol{U}_t = [\boldsymbol{u}_t^{(1)}, \cdots, \mathbf{u}_t^{(p)}] \in \mathbb{C}^{N\times p}$ spans the p-dimensional subspace of $\boldsymbol{R}$, and $\boldsymbol{U}_0$ is an arbitrary initial matrix with the property of $\boldsymbol{U}_0^H\boldsymbol{U}_0 = \boldsymbol{I}_{p\times p}$. Similar to the decentralized 1-dimensional subspace estimate, the computation of (8) requires the access to the entire vector of observations $\boldsymbol{r}_t$ to compute $\boldsymbol{r}_t^H\boldsymbol{U}_{t-1}$. In addition, each node also needs to estimate all the elements in the Hermitian matrix $\boldsymbol{U}_{t-1}^H\boldsymbol{U}_{t-1}$. Hence, there are total number of $(p^2 + p)/2 + p$ parameters which need to be distributedly computed via the average consensus protocol. *Algorithm 2* summarizes the implementation of the decentralized p-dimensional subspace tracking. Here, $\boldsymbol{U}_{0,0}$ is an arbitrary initial matrix with the property that $\boldsymbol{U}_{0,0}^H\boldsymbol{U}_{0,0} = \boldsymbol{I}_{p\times p}$.

Algorithm 2. p-Dimensional Subspace Tracking (at the i^{th} node)

```
   input  : A sequence of $r_{i,t}$
   output: The $i^{\text{th}}$ row of the principal subspace $\boldsymbol{U}$
1  Initialization: $t = 0$;
2    $\boldsymbol{u}_{i,1:p} = 0,\ u_{i,i} = 1$;
3  Recursion:
4    foreach $t = 1 : T$ do
5        $\mathbf{ac}_{i,1:p} \leftarrow N \cdot \mathrm{AC}(r_{i,t}^{*}\boldsymbol{u}_{i,1:p})$;
6        $\boldsymbol{v}_{i,1:p} \leftarrow 2r_{i,t} \cdot \mathbf{ac}_{i,1:p}$;
7        $\boldsymbol{v}_{i,1:p} \leftarrow \boldsymbol{v}_{i,1:p} - N\boldsymbol{v}_{i,1:p} \cdot \mathrm{AC}(\boldsymbol{u}_{i,1:p}^{H}\boldsymbol{u}_{i,1:p})$;
8        $\boldsymbol{v}_{i,1:p} \leftarrow \boldsymbol{v}_{i,1:p} - \boldsymbol{u}_{i,1:p}(\mathbf{ac}_{i,1:p}^{H}\mathbf{ac}_{i,1:p})$;
9        $\boldsymbol{u}_{i,1:p} \leftarrow \boldsymbol{u}_{i,1:p} + \gamma_t \boldsymbol{v}_{i,1:p}$;
10   end
```

Stability Analysis. Without loss of generality, the mathematical model relating the centralized and the decentralized networks is as follows (t is omitted)

$$\boldsymbol{r}\boldsymbol{r}^H\boldsymbol{U} \rightarrow [N\boldsymbol{W}^k \circ (\boldsymbol{r}\boldsymbol{r}^H)]\boldsymbol{U} \ ; \tag{9}$$

$$\boldsymbol{r}\boldsymbol{r}^H\boldsymbol{U}\boldsymbol{U}^H\boldsymbol{U} \rightarrow [N\boldsymbol{W}^k \circ (N\boldsymbol{W}^k \circ \boldsymbol{r}\boldsymbol{r}^H)\boldsymbol{U}\boldsymbol{U}^H]\boldsymbol{U} \ ; \tag{10}$$

$$\boldsymbol{U}\boldsymbol{U}^H\boldsymbol{r}\boldsymbol{r}^H\boldsymbol{U} \rightarrow [N\boldsymbol{W}^k \circ (N\boldsymbol{W}^k \circ \boldsymbol{U}\boldsymbol{U}^H)\boldsymbol{r}\boldsymbol{r}^H]\boldsymbol{U} \ . \tag{11}$$

Therefore, the ODE of (8) can be expressed as

$$\frac{d\boldsymbol{U}}{dt} = 2\boldsymbol{R}\boldsymbol{U} - \boldsymbol{R}\boldsymbol{U}\boldsymbol{U}^H\boldsymbol{U} - \boldsymbol{U}\boldsymbol{U}^H\boldsymbol{R}\boldsymbol{U} + \tilde{\boldsymbol{\Delta}}_{\mathrm{ODE}}\boldsymbol{U} \ , \tag{12}$$

$$\text{where} \quad \tilde{\boldsymbol{\Delta}}_{\mathrm{ODE}} = N\big[2\tilde{\boldsymbol{W}}^k \circ \boldsymbol{R} - (\tilde{\boldsymbol{W}}^k \circ \boldsymbol{R})\boldsymbol{U}\boldsymbol{U}^H - \tilde{\boldsymbol{W}}^k \circ \boldsymbol{R}\boldsymbol{U}\boldsymbol{U}^H \tag{13}$$
$$-\tilde{\boldsymbol{W}}^k \circ \boldsymbol{U}\boldsymbol{U}^H\boldsymbol{R} - (\tilde{\boldsymbol{W}}^k \circ \boldsymbol{U}\boldsymbol{U}^H)\boldsymbol{R}\big] + \mathcal{O}(N^2\tilde{\boldsymbol{W}}^{2k}) \ .$$

$\tilde{\boldsymbol{\Delta}}_{\mathrm{ODE}}$ is the trajectory deviation from the centralized ODE of $\boldsymbol{U}$. Similar to the 1-dimensional case, we want to study the trajectory of the $\boldsymbol{U}^H\boldsymbol{U}$ in order to learn the stability of (12), the result of which is stated in Lemma 3.

Lemma 3 (Convergence). *The trajectory of $\boldsymbol{U}^H\boldsymbol{U}$ asymptotically converges to $\boldsymbol{I}_{p\times p}$ if the number of the average consensus iterations k is sufficiently large. Furthermore, the columns in $\boldsymbol{U}$ span the p-dimensional principal subspace of $\boldsymbol{R}$.*

Proof. See Appendix C. □

Numerical Results. The subspace estimate is defined to be the Euclidean norm of the estimated p-dimensional subspace projected onto the $(N-p)$-dimensional non-dominating subspace, $\|(\boldsymbol{U}_{\mathrm{true}}^{\perp})^H\boldsymbol{U}_{\mathrm{est}}\|^2$. Fig. 3 illustrates the performance numerically for a data matrix associated to a 2-D subspace. The source contains several *white noise* corrupted narrowband signals at 20dB SNR. There are 10 sensors in the network and $k = 5$.

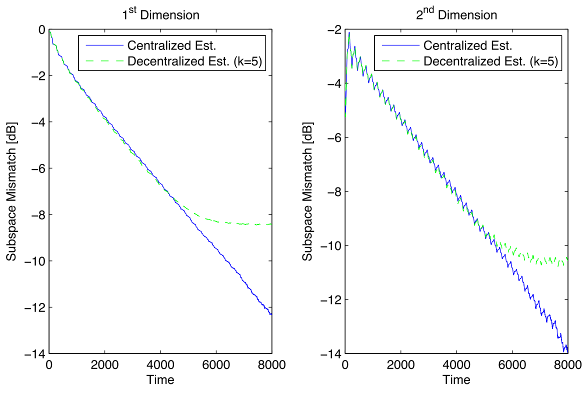


Fig. 3. Numerical results for the p-dimensional subspace tracking algorithm, p=2

Remark 2. A peculiar aspect of our protocol is that each node obtains an estimate of a single component of the principal vectors. This information in itself bears value to the node, since the magnitude of this component represents a ranking of the relative proximity of the source to the specific node that can be used directly to manage network operations. A more rigorous approach to use the information to detect the source is discussed next.

3 Case Study : Decentralized Narrowband Signal Detection in a Sensor Field

Recently the Federal Communications Commission (FCC) has launched technical reviews of cognitive radio (CR) devices as a way of better utilizing the scarce spectrum resource. Such CRs are capable of sensing the spectral environment (e.g. the narrowband low power microphone signals in the TV band) and exploiting the unoccupied spectrum. One possible approach to improve the spectral estimation is by using a *distributed model*, in which the spectrum occupancy is determined by the joint detection of all the CRs. Several papers have written on this issue, e.g. [25,26]. Our idea here is to link more closely the detection of the active primary user to the classic array processing problem of detecting sources using a sensor array, albeit with a *networked* array in lieu of the traditional co-located array. We propose to use the spatio-temporal covariance matrix of the detected data to identify the presence of the low power narrowband signal.

3.1 Signal Model and Signal Subspace

Signal Model Suppose there are N sensors (CRs) in the network to measure the temporal data. The received signal samples by the n^{th} sensor is $\boldsymbol{r}_n = \xi_n \boldsymbol{s} + \boldsymbol{w}_n$, for $n = 1, \cdots, N$, where ξ_n is the signal attenuation and $\boldsymbol{r}_n \in \mathbb{C}^M$ is the received samples, while $\boldsymbol{s} \in \mathbb{C}^M$ is the narrowband signal and $\boldsymbol{w}_n \in \mathbb{C}^M$ denotes the additive white Gaussian noise (AWGN) at the sensor. By stacking the received samples into one vector as $\boldsymbol{r} = \left(\boldsymbol{r}_1^T, \cdots, \boldsymbol{r}_N^T\right)^T$, we have

$$\boldsymbol{r} = (\boldsymbol{\xi} \otimes \boldsymbol{I}_M) \cdot \boldsymbol{s} + \boldsymbol{w} \ , \tag{14}$$

where $\boldsymbol{\xi} = (\xi_1, \cdots, \xi_N)^T$ represents the steering vector of the spatially co-located sensors. Then, the spatio-temporal covariance of the received samples is $\boldsymbol{R} = \mathbb{E}\{\boldsymbol{r}\boldsymbol{r}^H\} = \boldsymbol{R}_\xi \otimes \boldsymbol{R}_s + \boldsymbol{R}_w$, where $\boldsymbol{R}_s \triangleq \mathbb{E}\{\boldsymbol{s}\boldsymbol{s}^H\}$ is the transmitted signal temporal covariance and $\boldsymbol{R}_w$ is the noise covariance and is usually spatio-temporally white.

By eigenvalue-decomposition, we get $\boldsymbol{R}_s \triangleq \boldsymbol{U}_s \boldsymbol{\Sigma}_s \boldsymbol{U}_s^H$ and $\boldsymbol{R}_\xi \triangleq \boldsymbol{U}_\xi \boldsymbol{\Sigma}_\xi \boldsymbol{U}_\xi^H$. Due to the fact that $\boldsymbol{R}_\xi$ is a rank-one matrix, we have the following decomposition for the spatio-temporal covariance $\boldsymbol{R}$ of the received samples

$$\boldsymbol{R} = \boldsymbol{U}_R \boldsymbol{\Sigma} \boldsymbol{U}_R^H \ , \tag{15}$$

$$\text{where} \quad \boldsymbol{U}_R = \boldsymbol{U}_\xi \otimes \boldsymbol{U}_s \quad \text{and} \qquad \boldsymbol{\Sigma} = \operatorname{diag}\left(\|\boldsymbol{\xi}\|^2 \boldsymbol{\Sigma}_s + \sigma^2 \boldsymbol{I}_M,\ \sigma^2 \boldsymbol{I}_{(N-1)M}\right) \ .$$

With N being sufficiently large, the eigenvalues (diagonal entries of $\boldsymbol{\Sigma}_s$) only peak at a few positions where the narrowband source signal sits. Thus, we can first determine the eigenvectors that correspond to the active components (i.e., the signal subspace) and perform detection using the signal subspace.

Signal Subspace Tracking via Gossiping. The spatio-temporal covariance of $\boldsymbol{r}$ in (14) is $\boldsymbol{R} = \mathbb{E}\{\boldsymbol{r}\boldsymbol{r}^H\} \in \mathbb{C}^{NM \times NM}$. To adaptively track its subspace, we apply the decentralized p-D subspace tracking algorithm with slight modifications. Since each sensor receives a vector of temporal data instead of a single observation at each time, the decentralization of $\boldsymbol{r}^H \boldsymbol{U}$ and $\boldsymbol{U}^H \boldsymbol{U}$ in (8) are

$$\boldsymbol{r}^H \boldsymbol{U} = \sum_{n=1}^{N} \boldsymbol{r}_n^H \boldsymbol{U}_n \leftarrow N \boldsymbol{W}^k \left[(\boldsymbol{r}_1^H \boldsymbol{U}_1)^T \cdots (\boldsymbol{r}_N^H \boldsymbol{U}_N)^T\right]^T \ ; \tag{16}$$

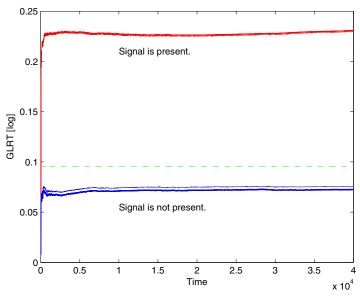
$$\left[\boldsymbol{U}^H \boldsymbol{U}\right]_{\ell,j} = \sum_{n=1}^{N} \left[\boldsymbol{U}_n^H \boldsymbol{U}_n\right]_{\ell,j} \leftarrow N \boldsymbol{W}^k \left[\left[\boldsymbol{U}_1^H \boldsymbol{U}_1\right]_{\ell,j}^T \cdots \left[\boldsymbol{U}_N^H \boldsymbol{U}_N\right]_{\ell,j}^T\right]^T \ , \tag{17}$$

where $\boldsymbol{U} = \left(\boldsymbol{U}_1^T \cdots \boldsymbol{U}_N^T\right)^T \in \mathbb{C}^{MN \times p}$, $\boldsymbol{U}_n \in \mathbb{C}^{N \times p}$ is a partial subspace estimate computed at the nth node and $[\ ]_{\ell,j}$ denotes the $\{\ell, j\}$ element of the matrix. (16, 17) indicate that the network wide consensus is achieved by processing blocks of data (i.e. $\boldsymbol{U}_n^H \boldsymbol{U}_n$) at each iteration. Even though the size of the covariance is larger, the consensus rate should not be affected because the network topology (or $\boldsymbol{W}$) stays the same. The analysis on the p-D subspace estimate still applies.

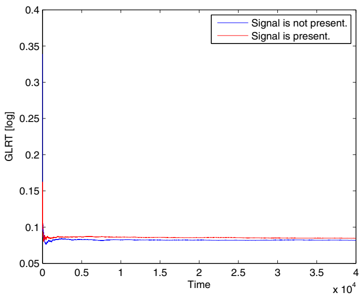
3.2 Signal Detection

Now given the available signal subspace, denoted as $\boldsymbol{U}_\parallel$, that is tracked among the nodes using the proposed algorithm, the signal detection problem becomes a binary hypothesis testing problem as follows

$$\mathcal{H}_0 : \boldsymbol{r} = \boldsymbol{w} \ , \ \ \mathcal{H}_1 : \boldsymbol{r} = \boldsymbol{U}_\parallel \boldsymbol{s}_U + \boldsymbol{w} \ . \tag{18}$$



(a) Ten-sensor network



(b) One sensor

Fig. 4. Performance of the decentralized narrowband signal detection

Notice that (18) is a classic estimation and detection problem of unknown signal's amplitude in Gaussian noise of unknown variance. The Generalized Likelihood Ratio Test (GLRT) [28] in this special case is as follows

$$\boldsymbol{L}_{\mathrm{GLR}}(\boldsymbol{r}) = \frac{P(\boldsymbol{r}|\mathcal{H}_1, \hat{\boldsymbol{s}}_U, \hat{\sigma}_1^2)}{P(\boldsymbol{r}|\mathcal{H}_0, \hat{\sigma}_0^2)} = \frac{\hat{\sigma}_0^2}{\hat{\sigma}_1^2} \ . \tag{19}$$

The maximum likelihood estimates of the unknown parameters are $\hat{\boldsymbol{s}}_U = \boldsymbol{U}_{\|}^H \boldsymbol{r}$, $\hat{\sigma}_1^2 = \frac{\|\boldsymbol{r}-\boldsymbol{U}_{\|}\hat{\boldsymbol{s}}_U\|^2}{MN}$, $\hat{\sigma}_0^2 = \frac{\|\boldsymbol{r}\|^2}{MN}$. Hence the detection rule is $\boldsymbol{L}_{\mathrm{GLR}}(\boldsymbol{r}) = \boldsymbol{r}^H\boldsymbol{r}/(\boldsymbol{r}^H\boldsymbol{r} - \|\boldsymbol{r}^H\boldsymbol{U}_{\|}\|^2) \geq \chi$ if the signal is present. Consider now the binary hypothesis testing problem with T consecutive observations, $\boldsymbol{r}_t$ with $1 \leq t \leq T$. The log-likelihood ratio for the binary hypothesis testing problem becomes,

$$\ln(\boldsymbol{L}_{\mathrm{GLR}}(\boldsymbol{r})) = \sum_{t=1}^{T} \ln\left(\frac{\boldsymbol{r}_t^H\boldsymbol{r}_t}{\boldsymbol{r}_t^H\boldsymbol{r}_t - \|\boldsymbol{r}_t^H\boldsymbol{U}_{\|}\|^2}\right) \geq T\ln\chi \ . \tag{20}$$

Thus, to make a distributed detection based on (20), each node must be able to obtain the decentralized estimates of $\boldsymbol{r}^H\boldsymbol{r}$ and $\boldsymbol{r}^H\boldsymbol{U}_{\|}$ via gossiping and then compute the log-likelihood ratio in (20).

Fig. 4 illustrates the performance of the decentralized subspace tracking and detection algorithm in a sensor field. Fig. 4a displays the detection output in a 10-sensor network. The dashed line represents the threshold. Fig. 4b shows the detection output of a single sensor. By comparing the two, we observe that the separation between the GLRTs when the signal is present and it is not present is significantly larger in the 10-sensor network. Especially when the separation is as small as shown in Fig. 4b, not only it is difficult to set the threshold, but also the detection is prone to high false alarm rate and missed detection rate.

4 Conclusion

This paper proposed a decentralized subspace tracking algorithm based on Oja's learning rule [5,9]. The key idea is to enable an in-network computation algorithm via near-neighbor communication without setting routes to forward the data to the fusion center. Through the stability analysis, we showed that given the number of consensus iteration k is sufficiently large, the system is self-stablizing ($\boldsymbol{U}^H\boldsymbol{U} \rightarrow \boldsymbol{I}$) and furthermore, it converges to the subspace asymptotically. To motivate our study, we present a study case on cognitive radio system to distributedly detect the presence of a low power narrowband signal. By comparing the proposed decentralized detection algorithm with a simple one-sensor scenario, our algorithm shows a significant performance gain.

References

1. Scaglione, A., Pagliari, R., Krim, H.: The decentralized estimation of the sample covariance. Asilomar, Pacific Grove (2008)
2. Stewart, G.: Matrix algorithms. SIAM, Philadelphia (1998)
3. Xiao, L., Boyd, S.: Fast linear iterations for distributed averaging. System and Control Letters (2004)
4. Olfati-Saber, R., Fax, A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proc. of the IEEE 95(1), 215–233 (2007)
5. Oja, E.: A simplified neuron model as a principal component analyzer. Journal of Mathematical Biology (15), 267–273 (1982)
6. Karhunen, J.: Adaptive algorithm for estimating eigenvectors of correlation type matrices. In: Proc. ICASSP 1984, vol. 9, pp. 592–595 (March 1984)
7. Sayed, A.H., Lopes, C.G.: Adaptive processing over distributed networks. IEICE Trans. Fund. Electron. Comm. Comput. Sci. E90-A(8), 1504–1510 (2007)
8. Xiao, L., Boyd, S., Lall, S.: A scheme for robust distributed sensor fusion based on average consensus. In: Proc. 4^{th} IPSN, April 2005, pp. 63–70 (2005)
9. Manton, J., Mareels, I., Attallah, S.: An analysis of the fast subspace tracking algorithm Noja. In: Proc. ICASSP 2002, vol. 2, pp. 1101–1104 (2002)
10. Attallah, S., Abed-Meraim, K.: Fast algorithm for subspace tracking. IEEE Signal Processing Letters 8(7), 203–206 (2001)
11. Yang, B.: Projection approximation subspace tracking. IEEE Trans. on Sig. Proc. 43(1), 95–107 (1995)
12. Owsley, N.L.: Adaptive data orthogonalization. In: Proc. ICASSP, pp. 109–112 (1978)
13. Sharman, K.C.: Adaptive algorithms for estimating the complete covariance eigenstrcuture. In: Proc. IEEE ICASSP, pp. 1401–1404 (April 1986)
14. Bunch, J.R., Nielsen, C.P., Sorenson, D.: Rank-one modification of the symmetric eigenproblem. Numerische Mathematik 31, 31–48 (1978)
15. Sardellitti, S., Giona, M., Barbarossa, S.: Fast distributed consensus algorithms based on advection-diffusion processes. In: Sensor Array and Multichannel Signal Processing Workshop (SAM) 2008, July 21-23, pp. 266–270 (2008)
16. Barbarossa, S., Scutari, G.: Decentralized maximum likelihood estimation for sensor networks composed of nonlinearly coupled dynamical systems. IEEE Trans. on Sig. Proc. 55(7), Part 1, 3456–3470 (2007)
17. Kar, S., Moura, J.M.F., Ramanan, K.: Distributed parameter estimation in sensor networks: nonlinear observation models and imperfect communication. IEEE Trans. on Info. Theory (August 2008) (submitted to), arxiv.org/abs/0809.0009
18. Rabbat, M., Nowak, R.D.: Distributed optimization in sensor networks. In: Proc. 3^{th} IPSN 2004, Berkeley, CA, pp. 20–27 (April 2004)

19. Schizas, I.D., Giannakis, G.B., Roumeliotis, S.D., Ribeiro, A.: Consensus in Ad Hoc WSNs with noisy links - Part II: Distributed estimation and smoothing of random signals. IEEE Trans. on Sig. Proc. 56(4), 1650–1666 (2008)
20. Nedic, A., Ozdaglar, A., Parrilo, A.P.: Constrained consensus and optimization in multi-agent networks. LIDS Technical Report 2779, MIT, Lab. for Information and Decision Systems, IEEE Transactions on Automatic Control (2009) (to appear)
21. Olfati-Saber, R.: Distributed Kalman filter with embedded consensus filters. In: Proc. 44th IEEE Conf. Eur. Contr. Conf., Seville, Spain, pp. 8179–8184 (December 2005)
22. Veeravalli, V.V., Basar, T., Poor, H.V.: Minimax robust decentralized detection. IEEE Trans. Inform. Theory 40(1), 35–40 (1994)
23. Cattivelli. F., Sayed, A.H.: Diffusion LMS strategies for distributed estimation. IEEE Transactions on Signal Processing (2010) (to appear)
24. Penrose, M.: Random geometric graph. Oxford Studies in Probability (2003)
25. Chen, H.S., Gao, W., Daut, D.: Spectrum sensing for wireless microphone signals. In: 5th IEEE SECON Workshops 2008, p. 15 (June 2008)
26. Unnikrishnan, J., Veeravalli, V.V.: Cooperative sensing for primary detection in cognitive radio. IEEE Journal on Selected Topics in Signal Processing, Special Issue on Dynamic Spectrum Access 2(1), 18–27 (2008)
27. Kailath, T.: Linear systems. Prentice-Hall, Englewood Cliffs (1980)
28. Kay, S.M., Gabriel, J.R.: An invariance property of the generalized likelihood ratio test. IEEE Signal Proc. Letters 10, 352–355 (2003)
29. Yan, W.Y., Helmke, U., Moore, J.B.: Global analysis of Oja's flow for neural networks. IEEE Trans. Neural Networks, 674–683 (September 1994)

Appendix

A. Proof of Lemma 1

Proof. From the ODE expression in (6,7), the trajectory of $\boldsymbol{u}^H\boldsymbol{u}$ is

$$\begin{aligned}\frac{d\boldsymbol{u}^H\boldsymbol{u}}{dt} &= \boldsymbol{u}^H\frac{d\boldsymbol{u}}{dt}+\frac{d\boldsymbol{u}^H}{dt}\boldsymbol{u} = 2(\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u})\left(1-\boldsymbol{u}^H\boldsymbol{u}+\delta_{\mathrm{ODE}}\right) \ , \\ \text{where} \quad \delta_{\mathrm{ODE}} &= \frac{\boldsymbol{u}^H(\boldsymbol{\Delta}_{\mathrm{ODE}}+\boldsymbol{\Delta}_{\mathrm{ODE}}^H)\boldsymbol{u}}{2\ \boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}}\end{aligned} \tag{21}$$

δ_{ODE} represents the trajectory of the decentralized $\boldsymbol{u}^H\boldsymbol{u}$ deviating from the trajectory of centralized (i.e. $k=\infty$) estimate. From the preceding relation (21), the equilibrium point is $\boldsymbol{u}_{\mathrm{eq.}}^H\boldsymbol{u}_{\mathrm{eq.}}=1+\delta_{\mathrm{ODE}}$. Hence, studying the trajectory of $\boldsymbol{u}^H\boldsymbol{u}$ is equivalent to learning its ODE deviation δ_{ODE}.

Using the fact that $\hat{\boldsymbol{W}}$ is dominated by the eigenpair $(\lambda_{2,w},\boldsymbol{v}_2)$, then the approximation $\hat{\boldsymbol{W}}\approx\lambda_{2,w}\boldsymbol{v}_2\boldsymbol{v}_2^T$ holds. From the relation in (7), we have

$$\begin{aligned}\boldsymbol{u}^H\boldsymbol{\Delta}_{\mathrm{ODE}}\boldsymbol{u} &\approx N\lambda_{2,w}^k\boldsymbol{u}^H\left(\boldsymbol{v}_2\boldsymbol{v}_2^T\circ\boldsymbol{R}-\boldsymbol{v}_2\boldsymbol{v}_2^T\circ\boldsymbol{u}\boldsymbol{u}^H\boldsymbol{R}-(\boldsymbol{v}_2\boldsymbol{v}_2^T\circ\boldsymbol{u}\boldsymbol{u}^H)\boldsymbol{R}\right)\boldsymbol{u} \\ &= N\lambda_{2,w}^k\boldsymbol{u}^H\left(\boldsymbol{v}_2\boldsymbol{v}_2^T\circ\boldsymbol{R}-2\Re\{(\boldsymbol{v}_2\circ\boldsymbol{u})^H\boldsymbol{R}\boldsymbol{u}\}\mathrm{diag}(\boldsymbol{v}_2)\right)\boldsymbol{u} \ .\end{aligned}$$

Hence, plugging the preceding relation into (21) yields

$$\delta_{\mathrm{ODE}} = N\lambda_{2,w}^k\left(\frac{\boldsymbol{u}^H(\boldsymbol{v}_2\boldsymbol{v}_2^H\circ\boldsymbol{R})\boldsymbol{u}}{\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}}-\frac{2\boldsymbol{u}^H\Re\{(\boldsymbol{v}_2\circ\boldsymbol{u})^H\boldsymbol{R}\boldsymbol{u}\}\mathrm{diag}(\boldsymbol{v}_2)\boldsymbol{u}}{\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}}\right) \ .$$

Since the spectral radius of the Hadamard product is bounded by $\rho(\boldsymbol{v}_2\boldsymbol{v}_2^H \circ \boldsymbol{R}) \leq \rho(\boldsymbol{v}_2\boldsymbol{v}_2^H)\rho(\boldsymbol{R}) = \rho(\boldsymbol{R})$, then it can be easily derived that

$$\frac{\boldsymbol{u}^H(\boldsymbol{v}_2\boldsymbol{v}_2^H \circ \boldsymbol{R})\boldsymbol{u}}{\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}} \leq \max_{\boldsymbol{u}} \frac{\boldsymbol{u}^H(\boldsymbol{v}_2\boldsymbol{v}_2^H \circ \boldsymbol{R})\boldsymbol{u}}{\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}} \leq 1. \tag{22}$$

Using the properties $(\boldsymbol{v} \circ \boldsymbol{u})^H\boldsymbol{u} \leq ||\boldsymbol{v}||_\infty \boldsymbol{u}^H\boldsymbol{u}$ and rayleigh quotient, we obtain

$$\begin{aligned} & \frac{2\boldsymbol{u}^H \Re\{(\boldsymbol{v}_2 \circ \boldsymbol{u})^H \boldsymbol{R}\boldsymbol{u}\}\text{diag}(\boldsymbol{v}_2)\boldsymbol{u}}{\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}} \\ \leq & \max_{\boldsymbol{u}} \frac{2\Re\{\boldsymbol{u}^H\boldsymbol{R}^{-1/2}\text{diag}(\boldsymbol{v}_2)\boldsymbol{R}^{1/2}\boldsymbol{u}\}}{\boldsymbol{u}^H\boldsymbol{u}} ||\boldsymbol{v}_2||_\infty \boldsymbol{u}^H\boldsymbol{u} \\ = & 2||\boldsymbol{v}_2||_\infty \boldsymbol{u}^H\boldsymbol{u}\ \lambda_{max}\{\text{diag}(\boldsymbol{v}_2)\} \\ = & 2||\boldsymbol{v}_2||_\infty^2 \boldsymbol{u}^H\boldsymbol{u}\ . \end{aligned} \tag{23}$$

Using the results from (22, 23), we get an upper bound on δ_{ODE},

$$|\delta_{\text{ODE}}| \leq N\lambda_{2,w}^k(1 + 2||\boldsymbol{v}_2||_\infty^2 \boldsymbol{u}^H\boldsymbol{u})\ ,$$

where $||\boldsymbol{v}_2||_\infty < 1$ and $\lambda_{2,w}^k \to 0$. Hence, $\boldsymbol{u}_{\text{eq.}}^H\boldsymbol{u}_{\text{eq.}}$ approaches to 1 on the order of $N\lambda_{2,w}^k$. For k sufficiently large, specifically, $\{k \in \mathbb{N}_+ | \lambda_{2,w}^k < 1/(2||\boldsymbol{v}_2||_\infty^2 N)$, then $\boldsymbol{u}_{\text{eq.}}^H\boldsymbol{u}_{\text{eq.}} \in \left(\frac{1-N\lambda_{2,w}^k}{1+2||\boldsymbol{v}_2||_\infty^2 N\lambda_{2,w}^k}, \frac{1+N\lambda_{2,w}^k}{1-2||\boldsymbol{v}_2||_\infty^2 N\lambda_{2,w}^k}\right) \xrightarrow{k\to\infty} 1$ asymptotically. □

B. Proof of Lemma 2

Proof. $\hat{\boldsymbol{W}}$ is approximated by the 2nd largest eigenpair of $\boldsymbol{W}$. Let the EVD of $\mathbf{R}$ be $\mathbf{R} = \sum_{i=1}^N \lambda_i \mathbf{u}_i\mathbf{u}_i^H\mathbf{u}$ for $\lambda_1 > \lambda_2 > \cdots > \lambda_N$. Then the ODE of $\mathbf{u}$ in (6) is

$$\frac{d\boldsymbol{u}}{dt} \approx \Bigg[\boldsymbol{R} - \boldsymbol{u}\boldsymbol{u}^H\boldsymbol{R} + N\lambda_{2,w}^k \underbrace{(\boldsymbol{v}_2\boldsymbol{v}_2^H \circ \boldsymbol{R})}_{:=\hat{\boldsymbol{R}}_v} - 2N\lambda_{2,w}^k \underbrace{\Re\{(\boldsymbol{v}_2 \circ \boldsymbol{u})^H\boldsymbol{R}\boldsymbol{u}\}}_{:=\beta} \text{diag}(\boldsymbol{v}_2)\Bigg]\mathbf{u}\ .$$

Decompose the 1-D subspace estimate $\boldsymbol{u}$ into $\boldsymbol{u} = \sum_{i=1}^N \alpha_i\boldsymbol{u}_i$, where $\boldsymbol{u}_i$'s are orthonormal. The ODE in the direction of $\boldsymbol{u}_i$ is computed independently,

$$\frac{d\alpha_i}{dt} = \lambda_i\alpha_i - \alpha_i\left(\boldsymbol{u}^H\boldsymbol{R}\boldsymbol{u}\right) + N\lambda_{2,w}^k(\boldsymbol{u}_i^H\hat{\boldsymbol{R}}_v\boldsymbol{u}_i)\alpha_i - 2N\lambda_{2,k}^k\beta(\boldsymbol{u}_i^H\text{diag}(\boldsymbol{v}_2)\boldsymbol{u}_i)\alpha_i\ .$$

Let $\zeta_i = \alpha_i/\alpha_1$ be the ratio of the coefficients of $\boldsymbol{u}$ along the $\boldsymbol{u}_i$ and $\boldsymbol{u}_1$ directions, and $\sigma = \boldsymbol{u}_i^H\hat{\boldsymbol{R}}_v\boldsymbol{u}_i - \boldsymbol{u}_1^H\hat{\boldsymbol{R}}_v\boldsymbol{u}_1 + 2\beta(\boldsymbol{u}_1^H\text{diag}(\boldsymbol{v}_2)\boldsymbol{u}_1 - \boldsymbol{u}_i^H\text{diag}(\boldsymbol{v}_2)\boldsymbol{u}_i)$,

$$\frac{d\zeta_i}{dt} = \mathbb{E}\left(\frac{1}{\alpha_1^2}(\frac{d\alpha_i}{dt}\alpha_1 - \frac{d\alpha_1}{dt}\alpha_i)\right) = -\zeta_i\left(\lambda_1 - \lambda_i - N\lambda_{2,w}^k\sigma\right) \tag{24}$$

with the solution $\zeta_i(t) \leq \zeta_m(t_0) \cdot e^{-\left(\lambda_1 - \lambda_i - N\lambda_{2,w}^k\sigma\right)}$, where λ_i is the eigenvalue of $\boldsymbol{R}$ associated to $\boldsymbol{u}_i$. Since σ is on the order of $||\boldsymbol{v}_2||_\infty^2\lambda_1$, we have $\left(\lambda_1 - \lambda_i - N\lambda_{2,w}^k\alpha\right) > 0$ for k sufficiently large. Lemma 1 proves $\boldsymbol{u}^H\boldsymbol{u} = 1 + \mathcal{O}(N\lambda_{2,w}^k)$, then $\lim_{t\to\infty}\sum_{j=1}^N \alpha_j^2(t) = 1 + \mathcal{O}(N\lambda_{2,w}^k)$. Hence the convergence of $\zeta_i(t)$ in (24) for $i = 2, \cdots, p$ implies the convergence of α_i to zero $(i = 2, \cdots, N)$, $\lim_{t\to\infty}\alpha_1(t)^2 = 1 + \mathcal{O}(N\lambda_{2,w}^k)$, thus $\boldsymbol{u}_t = \alpha_1(t)\boldsymbol{u}_1$. This concludes the proof. □

C. Proof of Lemma 3

Proof. The trajectory of $\boldsymbol{U}^H\boldsymbol{U}$ can be expressed as

$$\begin{aligned}\frac{d\boldsymbol{U}^H\boldsymbol{U}}{dt} &= \boldsymbol{U}^H\frac{d\boldsymbol{U}}{dt} + \frac{d\boldsymbol{U}^H}{dt}\boldsymbol{U} \\ &= 2\boldsymbol{U}^H\boldsymbol{R}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{U}^H\boldsymbol{U} + \tilde{\boldsymbol{\delta}}_{ODE}) + 2(\boldsymbol{I} - \boldsymbol{U}^H\boldsymbol{U} + \tilde{\boldsymbol{\delta}}_{ODE}^H)\boldsymbol{U}^H\boldsymbol{R}\boldsymbol{U}\ .\end{aligned} \tag{25}$$

$$\text{where}\ \ \tilde{\boldsymbol{\delta}}_{\text{ODE}} = \frac{1}{2}(\boldsymbol{U}^H\boldsymbol{R}\boldsymbol{U})^{-1}(\boldsymbol{U}^H\tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{U}) = \frac{1}{2}(\boldsymbol{U}^H\boldsymbol{U})^{-1}(\boldsymbol{U}^H\boldsymbol{R}^{-1}\tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{U})\ .$$

$\tilde{\boldsymbol{\delta}}_{\text{ODE}}$ represents the deviation of $\boldsymbol{U}^H\boldsymbol{U}$ in the decentralized approximation from its centralized approximation. (25) implies that the trajectory of $\boldsymbol{U}^H\boldsymbol{U}$ approaches $(\boldsymbol{I} + \tilde{\boldsymbol{\delta}}_{\text{ODE}})$ asymptotically and the size of $\tilde{\boldsymbol{\delta}}_{\text{ODE}}$ is influenced by the network topology. The Frobenius norm of $\tilde{\boldsymbol{\delta}}_{\text{ODE}}$ should diminish as k increases. Apply the approximation $\hat{\boldsymbol{W}}^k \approx \lambda_{2,w}^k\boldsymbol{v}_2\boldsymbol{v}_2^H$ to (25), we obtain

$$\tilde{\boldsymbol{\Delta}}_{\text{ODE}} \approx N\lambda_{2,w}^k\left[(\boldsymbol{v}_2\boldsymbol{v}_2^H\circ\boldsymbol{R})(2\boldsymbol{I}-\boldsymbol{U}\boldsymbol{U}^H)-2\Re\{\boldsymbol{v}_2\boldsymbol{v}_2\circ\boldsymbol{R}\boldsymbol{U}\boldsymbol{U}^H\}-(\boldsymbol{v}_2\boldsymbol{v}_2^H\circ\boldsymbol{U}\boldsymbol{U}^H)\boldsymbol{R}\right].$$

Using the sub-multiplicative property, the Frobenius norms are bounded by

$$\begin{aligned}||\tilde{\boldsymbol{\Delta}}_{\text{ODE}}||_{\text{F}} &\leq N\lambda_{2,w}^k 2||\boldsymbol{v}_2||_\infty^2||\boldsymbol{R}||_{\text{F}}(1 + 2\text{tr}(\boldsymbol{U}^H\boldsymbol{U}))\ ; \\ ||\tilde{\boldsymbol{\delta}}_{\text{ODE}}||_{\text{F}} &\leq \frac{1}{2}||(\boldsymbol{U}^H\boldsymbol{U})^{-1}\boldsymbol{U}^H||_{\text{F}}||\boldsymbol{R}^{-1}||_{\text{F}}||\tilde{\boldsymbol{\Delta}}_{\text{ODE}}||_{\text{F}}||\boldsymbol{U}||_{\text{F}} \\ &= N\lambda_{2,w}^k||\boldsymbol{v}_2||_\infty^2\underbrace{\text{tr}((\boldsymbol{U}^H\boldsymbol{U})^{-1})\text{tr}(\boldsymbol{U}^H\boldsymbol{U})}_{\delta_u}\underbrace{||\boldsymbol{R}^{-1}||_{\text{F}}||\boldsymbol{R}||_{\text{F}}}_{\delta_r}[1+2\text{tr}(\boldsymbol{U}^H\boldsymbol{U})] \\ &= N\lambda_{2,w}^k||\mathbf{v}_2||_\infty^2\delta_u\delta_r(1 + 2\text{tr}(\boldsymbol{U}^H\boldsymbol{U}))\ .\end{aligned}$$

The preceding inequality implies the effect of $\tilde{\boldsymbol{\delta}}_{\text{ODE}}$ diminishes as k increases. Hence, from (25), the equilibrium point is $\mathbf{U}^H\mathbf{U} \xrightarrow{k\to\infty} \mathbf{I}_{p\times p}$, then

$$\begin{aligned}\tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{U} &\approx N\lambda_{2,w}^k(\boldsymbol{v}_2\boldsymbol{v}_2^H\circ\boldsymbol{R} - 2\Re\{\boldsymbol{v}_2\boldsymbol{v}_2\circ\boldsymbol{R}\boldsymbol{U}\boldsymbol{U}^H\} - (\boldsymbol{v}_2\boldsymbol{v}_2^H\circ\boldsymbol{U}\boldsymbol{U}^H)\boldsymbol{R})\boldsymbol{U} \\ \frac{d\boldsymbol{U}}{dt} &= \boldsymbol{R}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{U}^H\boldsymbol{U}) + (\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^H + \tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{R}^{-1})\boldsymbol{R}\boldsymbol{U}\ .\end{aligned}$$

Using the approximation $\text{tr}(\boldsymbol{U}^H\boldsymbol{U}) \approx \text{tr}(\boldsymbol{I}) = p$, the norm $||\tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{R}^{-1}||_{\text{F}}$ is

$$\begin{aligned}||\tilde{\boldsymbol{\Delta}}_{\text{ODE}}\boldsymbol{R}^{-1}||_{\text{F}} &\leq N\lambda_{2,w}^k||\boldsymbol{v}_2||_\infty^2\left(||\boldsymbol{R}||_{\text{F}}||\boldsymbol{R}^{-1}||_{\text{F}}[1 + \text{tr}(\boldsymbol{U}^H\boldsymbol{U})] + \text{tr}(\boldsymbol{U}^H\boldsymbol{U})\right) \\ &= N\lambda_{2,w}^k||\boldsymbol{v}_2||_\infty^2\left(\delta_r(1+p) + p\right)\end{aligned}$$

Hence the ODE of $\boldsymbol{U}$ reduces to $d\boldsymbol{U}/dt = \boldsymbol{R}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{U}^H\boldsymbol{U}) + (\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^H + \mathcal{O}(N\lambda_{2,w}^k\boldsymbol{J}))\boldsymbol{R}\boldsymbol{U}$. For k large enough, $d\boldsymbol{U}/dt$ reduces to the Oja's flow, which converges to the principal subspace of $\boldsymbol{R}$ [29]. This completes the proof. □

Building $(1-\epsilon)$ Dominating Sets Partition as Backbones in Wireless Sensor Networks Using Distributed Graph Coloring

Dhia Mahjoub and David W. Matula

Bobby B. Lyle School of Engineering
Southern Methodist University
Dallas, TX 75275-0122, USA
{dmahjoub,matula}@lyle.smu.edu

Abstract. We recently proposed in [19,20] to use sequential graph coloring as a systematic algorithmic method to build $(1-\epsilon)$ dominating sets partition in Wireless Sensor Networks (WSN) modeled as Random Geometric Graphs (RGG). The resulting partition of the network into dominating and almost dominating sets can be used as a series of rotating backbones in a WSN to prolong the network lifetime for the benefit of various applications. Graph coloring algorithms in RGGs offer proven constant approximation guarantees on the chromatic number. In this paper, we demonstrate that by combining a local vertex ordering with the greedy color selection strategy, we can in practice, minimize the number of colors used to color an RGG within a very narrow window of the chromatic number and concurrently also obtain a domatic partition size within a competitive factor of the domatic number. We also show that the minimal number of colors results in the first $(\delta + 1)$ color classes being provably dense enough to form independent sets that are $(1-\epsilon)$ dominating. The resulting first $(\delta + 1)$ independent sets, where δ is the minimum degree of the graph, are shown to cover typically over 99% of the nodes (e.g. $\epsilon < 0.01$), with at least 20% being fully dominating. These independent sets are subsequently made connected through virtual links using localized proximity rules to constitute planar connected backbones. The novelty of this paper is that we extend our recent work in [20] into the distributed setting and present an extensive experimental evaluation of known distributed coloring algorithms to answer the $(1-\epsilon)$ dominating sets partition problem. These algorithms are both topology and geometry-based and yield $O(1)$ times the chromatic number. They are also shown to be inherently localized with running times in $O(\Delta)$ where Δ is the maximum degree of the graph.

Keywords: Domatic partition problem, $(1-\epsilon)$ dominating sets partition, Wireless Sensor Network, Graph coloring, Distributed Algorithm.

1 Introduction

In a random dense deployment of sensor networks, we can take advantage of the redundancy and physical proximity of nodes and require that only a subset of them stay active at one time to fulfill the application's objectives (e.g. coverage, data gathering, monitoring) while the rest of the nodes stay in a sleep mode to conserve their energy [27]. For this approach to be efficient, different subsets of active nodes should be rotated successively. In a wireless sensor network, the concept of constructing a collection of disjoint dominating sets whose activity can be duty-cycled/rotated/scheduled is becoming increasingly as attractive as building a single minimum dominating set. In fact, having several disjoint dominating sets at the disposal of the sensor application can offer better fault-tolerance, load-balancing, and scalability as well as prolonged network lifetime. It also raises the possibility of catering to several quality of service requirements in terms of coverage accuracy, or traffic priorities.

1.1 Preliminaries

In this paper, we adopt the Unit Disk Graph (UDG) and Random Geometric Graph (RGG) models as defined in [20] to represent a wireless sensor network. Simply, random geometric graphs induce a probability distribution on unit disk graphs [6]. In our work, we use properties and approximation results stemming from both models. In general, the network is abstracted as an undirected graph $G = (V, E)$. We denote the number of nodes by $n = |V|$; the maximum degree is Δ, the minimum degree is δ, and average degree is $\overline{d}$. The distributed computation model we adopt is the standard synchronous message passing model, where time is split into discrete rounds. In each round, every node can perform some local computations, send a message to each neighbor, and receive messages from all neighbors. We assume that nodes exchange short messages of size $O(\log n)$ bits and a message sent in round R arrives to its neighboring destination(s) before the next round $R+1$ starts. All nodes start a computation synchronously and the time complexity of an algorithm is the number of rounds from the start until the last node terminates [30]. We also define a localized algorithm as an algorithm where each node operates solely on information that is available within a constant neighborhood of the node, typically the 1-hop neighborhood [17].

1.2 Related Work

Several recent works advocated the aforementioned strategy by putting it in the context of the domatic partition (DP) problem: an NP-hard graph theoretical problem whose objective is to find the largest number of disjoint dominating sets [20,15,25,27,28,19]. This number is upper bounded by $\delta + 1$. In [27] the authors define the maximum cluster-lifetime problem and propose a distributed approximation algorithm that finds a number of disjoint dominating sets within $O(\log n)$ of the optimal solution in arbitrary graphs. In [28], the authors propose the first centralized and distributed constant factor geometry-aware approximation algorithms to the domatic partition problem in Unit Disk Graphs. In [26], the authors propose a complex distributed topology-based solution to the connected domatic partition (CDP) problem in UDGs and describe a simple mechanism to rotate between the obtained disjoint connected dominating sets. Overall, the prior work on the subject consists in approximation algorithms both simple and intricate that address several variations of the domatic partition problem. Some works proposed to relax the node disjointness, where a node can participate in more than one dominating set of the partition [14], or to relax the 1-hop domination constraint by considering a set of vertices S dominating if any node in $V \setminus S$ is within k hops of S [31]. Algorithmically, some other works restricted the DP problem by focusing on building a connected domatic partition (CDP) of the network [26], which is then upper bounded by the connectivity of the graph $\kappa(G)$ rather than $\delta(G)+1$ or to require redundant domination (coverage) where any node in the network should be dominated by at least k backbones [27]. Algorithmically, graph coloring has been used to build a single small connected dominating set to serve as a backbone for network layer routing [18,29] and also to construct a series of dominating sets for MAC layer scheduling [4,16]. The current authors in [19,20] showed that simple topology-based graph coloring can, in practice, not only solve the domatic partition problem in RGGs with a competitive performance ratio but also provide up to $(\delta+1)$ disjoint independent sets that are $(1-\epsilon)$ dominating, where $\epsilon < 0.01$ on a large range of experimented graphs. In this work, we carry the study of [20] further by proposing a practical solution to the distributed $(1-\epsilon)$ dominating sets partition problem that is based on simple and localized hence scalable graph coloring algorithms.

1.3 Our Contributions and Outline

In light of the existing literature, the question to ask is what conditions can be relaxed in the DP problem such that we can still have a practical and useful model. In this work, we relax the objective that all $(\delta + 1)$ backbones cover all nodes. We then experiment with fast and efficient localized coloring algorithms to construct disjoint independent $(1 - \epsilon)$ dominating sets. These sets are further made connected, by distinguishing selected two-hop virtual links, and planar through localized rules and thus they can serve as virtual backbones with desirable properties for the benefit of coverage and data dissemination in wireless sensor networks. The aim of this work is to empirically study several existing vertex coloring algorithms, both topology and geometry locally aware in the distributed setting. We show that our experimental results are consistent with the time complexity lower and upper bounds of distributed coloring algorithms from [11]. Our primary contribution is to translate these good results on the coloring performance ratio and running time into a systematic method to build $(\delta + 1)$ $(1 - \epsilon)$ disjoint dominating backbones in wireless sensor networks modeled as random geometric graphs with minimum degree δ. Another contribution is that we empirically attain a competitive approximation factor comparable to the best current distributed approximation algorithm to the domatic partition problem [28]. We finally provide extensive experimental results on the localized coloring algorithms implemented in our study with regard to the coloring performance ratio, running time (number of rounds), domatic partition performance ratio, quality of coverage of the first $(\delta + 1)$ backbones as well as the quality of triangulation of a single backbone.

The remainder of the paper is organized as follows. Section 2 explains our distributed algorithmic approach. Section 3 provides a theoretical analysis of the algorithm and Section 4 describes the implementation and the experimental results we obtained. We conclude and propose future work in Section 5.

2 Our Backbone Selection Algorithm

We propose a localized backbone selection algorithm to build a collection of $(\delta+1)$ disjoint fully and nearly dominating sets. The algorithm consists in two phases: The first phase is a generic coloring phase where each node locally acquires the smallest color that does not conflict with its neighbors. Color conflicts are resolved differently depending on the coloring heuristic chosen. The coloring heuristics we study in this paper are: Trivial Greedy [11], Largest First [11], Lexicographic [5], 3Cliques-Last [5] and randomized selection $\Delta + 1$ coloring [11]. The coloring produces k color classes, i.e. k independent sets of vertices each with the same color that are fully or nearly dominating. In the second phase, for each independent vertex set, we build backbone links by using relay vertices between every two independent vertices within distance 2. Then to ensure virtual backbone planarity, we utilize the Gabriel Graph condition [24,20] on each backbone virtual link. The Gabriel graph is locally definable [20,2], is not a distance spanner but is an efficient energy spanner [2,10]. In our work, planarity is enforced on backbone virtual links, where a virtual link $\overline{uv}$ consists of two communication links in the original graph connecting two independent nodes u and v (with the same color) via a common relay node w. For future work, we propose to minimize the sum $d(u,w)+d(w,v)$ over candidate relay nodes so that u, w and v are nearly co-linear and so that the collection of physical links (edges in the original graph) through the relay nodes also form a planar graph (with no crossing edges). We defer the thorough investigation of this problem to future work. Notice that any efficient locally-definable planar proximity graph can be applied on the generated backbones such as the *Relative Neighborhood Graph (RNG)* or *Restricted Delauney Graph (RDG)*. Moreover, let's consider the special class of $G(n,r)$ known as well-distributed geometric graphs [2] where if $r > \sqrt{\frac{32}{3}} r_{con}$ ($r_{con} = O(\sqrt{\frac{log\ n}{n}})$ to ensure connectivity of $G(n,r)$ w.h.p) then any convex area of size at least $\frac{3\pi}{32}r^2$ (in the unit square) has at least one node in it. In these graphs, the nodes are evenly distributed across the unit square and do not contain large "holes": empty convex regions with area larger than $\frac{3\pi}{32}r^2$ [2]. Therefore, we might get a better stretch factor for the Gabriel graph than the $\theta(\sqrt{n})$ worst case [2]. We defer the study of the quality of *Gabriel graph*-based backbones in well-distributed RGGs to future work.

2.1 First Phase: Graph Coloring

Owing to the geometric structure of UDGs and consequently RGGs, it is possible to approximate the chromatic number $\chi(G)$ within a constant factor. In fact, any sequential greedy coloring is a 5-approximation algorithm of $\chi(G)$ in UDGs

[20] and the "lexicographic" ordering and Smallest Last are 3-approximation algorithms [5,20]. In the distributed setting, lexicographic ordering can be implemented locally if nodes are aware of their geometric locations [5]. Recently, the authors of [5] proposed the first distributed, geometry-oblivious, 3-approximation coloring algorithm on UDGs.

The details of our proposed first phase are given in Algorithm 1 which consists in a generic coloring phase executed by every node u. The algorithm is adapted from [8] with wake-up probability p=1. All the localized coloring algorithms we studied in this paper with the exception of 3Cliques-Last [5] apply Algorithm 1, and they differ in the way color conflicts between neighboring nodes are resolved.

Algorithm 1. Local algorithm for coloring $G(n, r)$

```
1 //Algorithm is executed independently by each node $u$;
2 Wake up: Enter the coloring round with probability $p$.
3 Pick a color: Select a tentative color $c$ that is the smallest color not taken
  by any of $u$'s neighbors in previous rounds.
4 Resolve conflicts: If there is an uncolored neighbor $v$ that has precedence
  in selecting its color, then $u$ loses color and tries in next round, else
5 Terminate: If $u$'s selected color is final, then become colored and exit the
  coloring algorithm.
```

The five coloring heuristics investigated are:
1. Trivial Greedy: We denote this algorithm by GCOL [11]. In round i, a node chooses a color then checks with its neighbors for a color conflict on the color it chose. As it checks its neighbors one by one, it loses the color at the first occurence of a conflict with a neighbor (who also may lose its color) and tries again in round $i + 1$.
2. Largest First: Denoted by LFCOL [11], where vertices with larger degree have precedence in keeping their selected colors.
3. Lexicographic: Denoted by LEXICO, where vertices with lower x coordinates have precedence in keeping their selected colors. Ties on x are broken based on the y coordinates [5].
4. 3Cliques-Last: Denoted by 3CL-LAST. This algorithm introduced in [5] uses the property of small local neighborhood in UDG. If a node has a neighborhood size of at most $3\omega(u) - 3$, where $\omega(u)$ is the size of the largest clique that u is part of, then u is said to have a small neighborhood. Nodes with small neighborhoods will pick their colors after their neighbors. The authors in [5] use an $O(n^3)$ heuristic to approximate $\omega(u)$ where n is the size of the graph. In our implementation of 3CL-LAST, we use Smallest Last [22,23], which runs faster in $O(n^2)$ time and offers the interesting property of finding a terminal large complete subgraph that often can be confirmed to be a maximum clique by Smallest Last-coloring of the neighborhood induced subgraph of u [20].
5. Δ+1 coloring: A node selects a color uniformly at random from a color palette $\{1..\Delta + 1\}$ which overrides step 3 in Algorithm 1.

2.2 Second Phase: Preparing the Backbone

This phase consists in applying two locally-definable procedures: a 2-Hop rule that identifies relay nodes between near-by independent nodes and a Gabriel graph rule that enforces backbone virtual links to be planar.

Algorithm 2. Local algorithm for preparing backbones

```
foreach independent set $S_c$ of color $c$ where $c \in [1, \delta+1]$ do
    //Algorithm is executed independently by each node $u$;
    foreach node $u$ in $S_c$ do
        foreach node $w$ in $N(u)$ do
            build table $T^c(u) = \{v | v \in N(w), v \neq u, color(u) = color(v)\}$
        //after $T^c(u)$ is built
        foreach node $v$ in $T^c(u)$ do
            foreach node $w$ in $T^c(u) | w \neq v$ do
                if $d(u,v)^2 < [d(u,w)^2 + d(v,w)^2]$ then
                    $T^c(u) = T^c(u) \setminus \{v\}$
                break
```

In each one of the first $(\delta+1)$ independent sets, every node u first requests from its neighbors $N(u)$ the list of their respective neighbors that have the same color as u and creates its table $T^c(u)$. $T^c(u)$ represents the set of distance-2 backbone neighbors v that u can reach through a common neighbor (relay node). u is virtually adjacent to v via a link $\overline{uv}$. Second, u checks that no two links $\overline{uv}$ cross by enforcing that no third backbone node w resides inside the disk of diameter $\overline{uv}$ (Gabriel rule [24]). This rule yields a planar virtual backbone which remains connected if the set of distance-2 backbone neighbors provides a connected graph. In each entry of $T^c(u)$, node u stores a distance-2 backbone neighbor v and the associated set of relay nodes denoted R_{uv} that u can use to reach v. The size of R_{uv} is upper bounded by Δ but in reality it is much smaller than that. The size of R_{uv} is proportional to the upper bound on the area of intersection of the two disks of radius r centered at u and v. That area is maximized when $d(u,v) = 1+\epsilon$ where ϵ is a very small value, i.e. u and v are barely independent, in which case the overlap area would have an expected number of relays of about 40% of the average degree.

2.3 Rotating the Backbones

Building a collection of $(\delta+1)$ disjoint $(1-\epsilon)$ dominating backbones allows any single backbone to be active during only a small fraction of time compared to the total time of system operation. The backbone rotation schedule can simply correspond to the activation of the first $(\delta+1)$ color classes one after another. For a period of time T (of the sensor network application), each one of the first $(\delta+1)$

backbones remains active for a period $\frac{T}{\delta+1}$. If it is possible to globally synchronize the nodes in the network, then backbone 1 can autonomously go active in the time interval $[0, t_1]$ and shuts down when time approaches t_1. Similarly, any backbone i is active in the interval $[\Sigma_{j=0}^{i-1} t_j, \Sigma_{j=0}^{i} t_j]$ [27]. When backbone i nears the expiration of its service period, nodes in backbone i locally send activation messages to their neighbors in backbone $i+1$. After it receives acknowledgement from backbone $i+1$, backbone i can go to sleep until the next service cycle. This ensures the backbone activation signalling stays local. It is possible to run the synchronized backbone rotation protocol on top of a more realistic asynchronous network by using "synchronizer" protocols [30] with the same time complexity but incurring a larger message complexity.

3 Complexity Analysis

3.1 Conjecture on the Approximation of the Domatic Partition

We define the span of graph G [5,12] (or inductiveness as defined in other references [9,7]) as $span(G) = \max_{H \subseteq G} \delta(H)$. Smallest Last achieves a minimum span and in UDGs, the minimum span is $3\omega(G) - 3$ thus Smallest Last gives a 3-approximation to the minimum coloring in UDGs. In this case, when a node u is being colored, it has no more than $3\omega(G) - 3$ neighbors colored before it. If we can also ensure this property in the distributed setting, then it means that when a node tries to select a color, it can have at most $3\omega(G) - 3$ neighbors conflicting with it on the color choice, and in the worst case, u might have to choose one color more than the size of that conflicting neighborhood. A further investigation of this property may lead to a proof on the expected number of disjoint dominating sets in the neighborhood of any node by using graph coloring in random geometric graphs.

3.2 Time and Message Complexities

In this section, we first give a general discussion on the time complexities and approximation ratios of distributed graph coloring algorithms, then we give an overview of the time and message complexities of our proposed backbone creation solution. Typically, $(\Delta + 1)$ coloring algorithms have time complexity of the form $O(f(\Delta) + \log^* n)$ [17], but most of these algorithms don't specifically attempt to economize the number of colors and focus mainly on speed. A simple observation is that fast coloring algorithms can take advantage of the generosity of the $(\Delta+1)$ upper bound on the chromatic number which practically decreases the chances of color conflicts as vertices individually attempt to pick a color. Using $Greedy - Color$ (i.e. a node always picks the smallest color not causing any conflicts with its neighbors) increases color conflicts and slows down the algorithm (as it attempts to resolve color conflicts). However, in [11], the authors point out that $Greedy - Color$ has the much desired local minimality property described by Grundy, i.e. no single vertex may have its color value decreased without affecting the color of some other (neighboring) vertex. Therefore, using

$Greedy-Color$ generally leads to fewer colors used, and potentially denser earlier color classes. In practice, we obtain that the first $(\delta+1)$ color classes are all practically fully and nearly dominating sets.
More generally, $(\Delta+1)$ coloring can run very fast in $O(\log n)$ or $O(\log^* n)$. GCOL and LFCOL are refinements of $(\Delta+1)$ coloring of a graph G and therefore are not easier in computational time than $(\Delta+1)$ coloring [11]. GCOL has a lower bound of $\Omega(\frac{log\ n}{log\ log\ n})$ and an upper bound of $O(\Delta+\tau_{COL})$ [11] where τ_{COL} is a known upper bound on the time complexity of $(\Delta+1)$ coloring, typically $O(\log {}^*n)$ [32]. LFCOL has a lower bound of $\Omega(\Delta)$ and an upper bound of $O(\Delta.\tau_{COL})$. In this paper, we assume both GCOL and LFCOL run in $O(\Delta)$ and both LEXICO and 3CL-LAST run in $O(n)$ in the worst case.
Notice that GCOL and LFCOL are 5-approximation algorithms for coloring UDGs [3] (and RGGs). LEXICO and 3CL-LAST are 3-approximation algorithms for coloring UDGs and since $\omega(G) \leq \chi(G)$ and $\Delta(G) \leq 6\omega(G)-6$ in UDGs then any $\Delta+1$ coloring is a 6-approximation algorithm for coloring UDGs [5,21].

In our proposed algorithm, in order to initially collect complete one-hop neighborhoods, every node broadcasts its status to all neighbors (ID and x,y coordinates), which requires one round of communication and $O(n)$ messages where the message size is $O(\log n)$. The coloring phase takes $O(\Delta)$ time in the typical case and $O(\Delta n)$ messages of size $O(\log n)$ each, and results in every node knowing its neighbors' final colors. For the backbone preparation phase defined in Algorithm 2, each backbone node explores its distance-2 neighborhood for peer backbone nodes reachable through common neighbors (relay nodes). This takes constant time assuming negligible local processing in synchronous systems [30], and $O(\Delta^2)$ messages per backbone node and the message size is $O(\Delta.\log n)$. We denote by $|MIS|$ the upper bound on the size of a backbone (relay nodes not included) which is the size of a maximum independent set in a random geometric graph $G(n,r)$ [19]. Since we focus on $(\delta+1)$ backbones then Algorithm 2 takes $O((\delta+1).|MIS|.\Delta^2)$ messages. Notice that each backbone node u requires $O(1)$ storage since the number of distance-2 independent neighbors of u that are colored the same is upper bounded by a constant (at most 23 [1]).

4 Experimental Results

In this section we discuss the experiments evaluating our solution. We implemented our own purpose-built simulator in $C\sharp.Net$ (Microsoft Visual Studio 2005) using Windows Forms and tested our method on randomly generated $G(n,r)$ graph instances. We evaluated the performance of our backbone selection solution with 5 different distributed coloring algorithms (GCOL, LFCOL, LEXICO, 3CL-LAST and $\Delta+1$). For all Figures of performance over random geometric graphs $G(n,r)$ in the unit square, the results are averaged over 20 instances with $n \in [50, 3200]$. In some cases, r is chosen such that the average degree $\overline{d} = 40$ and in other cases so that the minimum degree δ is either 20 or 40 for all samples.

4.1 Performance of the Coloring Algorithms

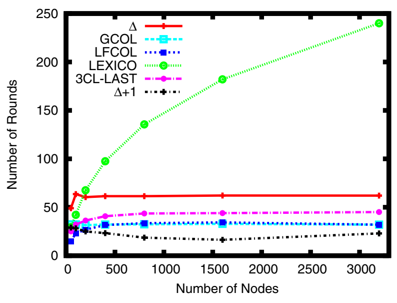
Figure 1a shows the running times of the coloring algorithms applied on graphs $G(n, r)$ in the unit square where $n \in [50, 3200]$ and $\overline{d} = 40$. We observe that GCOL, LFCOL, 3CL-LAST and $\Delta + 1$ coloring all have a running time in $O(\Delta)$ with a typical running time equal to $\overline{d}$. LEXICO shows a running time that grows with increasing n with a lower bound of Δ rounds when n is small. These experimental results are consistent with the theoretical bounds described above. Figure 1b shows the relative coloring speed between algorithms. The *time* axis represents the evolution of rounds scaled by the appropriate factor so that all algorithms run from t=0 to 100% of their actual running times. The *Colored nodes* axis shows the percentage of nodes colored. We observe that GCOL's coloring rate is linear, i.e. in every time window, a constant number of nodes gets colored throughout the network. The other algorithms also color at a linear rate but slower than GCOL. $\Delta + 1$ coloring colors around 60% of the network in the early rounds and is also the fastest since nodes choose colors randomly from their entire $\Delta + 1$ color palettes; therefore color conflicts are lower and the algorithm converges faster. Figure 1c plots the number of colors used by each algorithm as n grows with a constant $\delta = 20$. We use Smallest Last (SL) [22,23] as a benchmark for the coloring performance and also a lower bound on the size of the largest clique (lg-cl) that SL delivers as well. We observe that LFCOL and LEXICO have comparable performance, and so are GCOL and 3CL-LAST. $\Delta + 1$ coloring uses close to the upper bound number of colors $\Delta + 1$.

4.2 Quality Evaluation of a Single Dominating Backbone

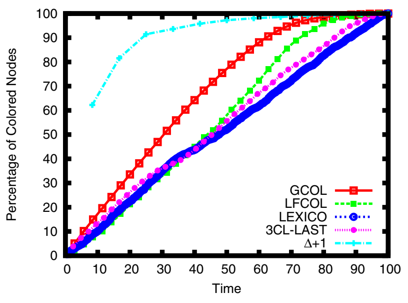
Figures 2a to 2d show sample triangulated dominating backbones obtained with each one of the coloring algorithms combined with the virtual links creation procedure (2-Hop rule and Gabriel rule). For a more detailed explanation of the color codification, refer to [20]. LEXICO has the best triangulated packing layout which also translates into a higher number of virtual links to connect the initial backbone independent nodes (see Figure 3a). The links in LEXICO are shorter with a median link length close to $1.2r$. The other coloring algorithms have similar behavior in their link length distributions. Figure 3b shows the $k - coverage$ performance of the sample backbones depicted that are obtained by each coloring algorithm. Figure 3b indicates the percentage of vertices that are at least covered by k backbone vertices for each one of Figures 2a to 2d for $k = 1, 2, 3, 4$. Intuitively, LEXICO offers the highest coverage for each k. For instance, more than 80% of the network nodes are covered by at least 2 backbone nodes and around 40% of the nodes are covered by at least 3 backbone nodes.

4.3 Quality Evaluation of $\delta + 1$ Backbones

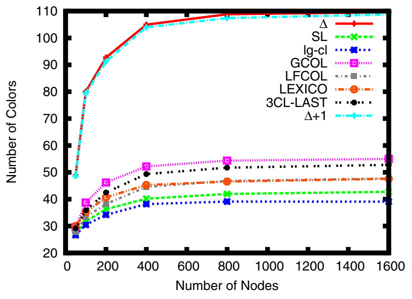
In Figures 4a and 4b, we plot the number of disjoint independent dominating sets obtained by the different coloring algorithms for graphs $G(n, r)$ where δ=20 and 40. Figures 4c and 4d plot the respective performance ratio as n grows. The



(a) Running time

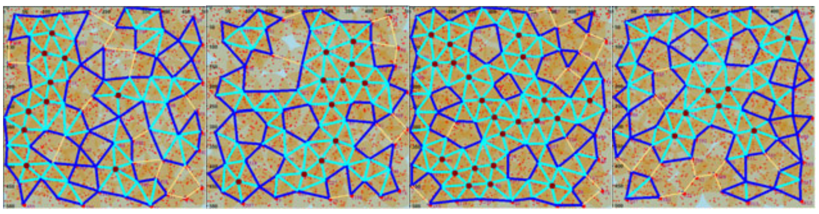


(b) Relative speed



(c) Minimum coloring

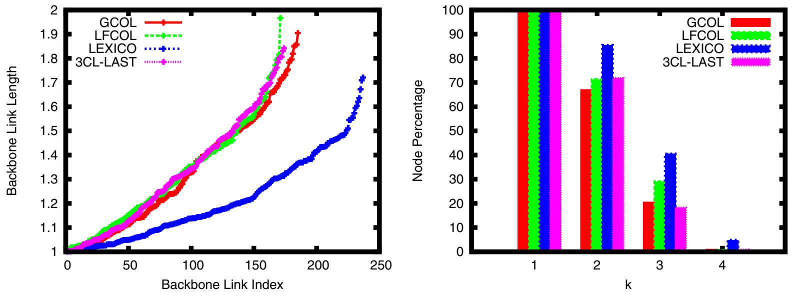
Fig. 1. Coloring algorithms performance



(a) GCOL (b) LFCOL (c) LEXICO (d) 3CL-LAST

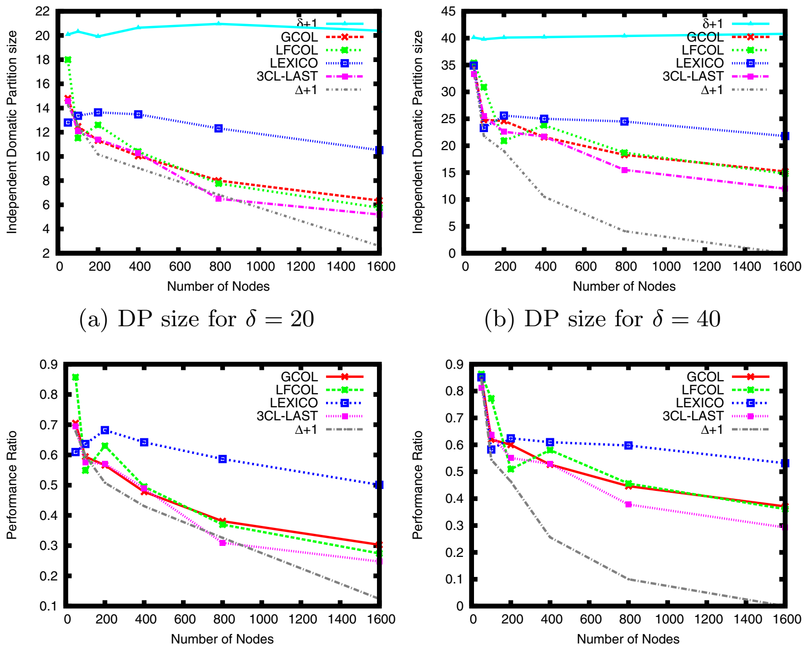
Fig. 2. Layout of a single dominating backbone

performance ratio of the domatic partition is simply the number of obtained disjoint independent dominating sets over the minimum degree δ plus 1. We observe that for most coloring algorithms the performance ratio starts high at 0.7 for small graphs and then decreases slowly as n grows to reach at least 0.2 for values of n continuing beyond 1600 (not shown in the plot). LEXICO is the least affected by the growing network size for a constant minimum degree. The performance ratio of LEXICO stabilizes at 0.5 for large n. Notice that the performance ratio for all coloring algorithms gets better for the same network sizes but with a higher δ. The performance of $\Delta+1$ coloring is shown for completeness, however, other than being the fastest to converge, this coloring procedure



(a) Link lengths (b) k-coverage

Fig. 3. Quality measures for a single dominating backbone

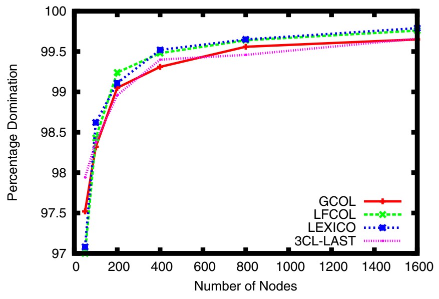


(a) DP size for $\delta = 20$ (b) DP size for $\delta = 40$

(c) Performance ratio for $\delta = 20$ (d) Performance ratio for $\delta = 40$

Fig. 4. Domatic partition size and performance ratio for variable density graphs

offers negligible advantages in domatic partition size as it degrades to 0 when the network becomes very large. On the other hand, we observe in Figure 5a that the $(1-\epsilon)$ domination of the first $\delta+1$ backbones increases as n grows. This happens concurrently as the strict domatic partition performance ratio decreases: as more independent sets lose in strict 100% domination (with growing n values), more almost dominating sets (with more than 99% domination) are obtained over the first $\delta+1$ color classes. Intuitively, for the same set size of $\delta+1$



(a) Domination of $\delta+1$ backbones

Fig. 5. Performance of the dominating and almost dominating $\delta+1$ backbones

backbones, when n grows, the domination quality is spread out over most of the backbones and rather than having a select few full dominating sets, we obtain more near 100% dominating sets. This implies that as more better quality (i.e. $(1-\epsilon)$) $\delta+1$ backbones are built in the network, any node in the network has more chances of being covered by most of the $\delta+1$ backbones [20]. Conversely, very few nodes in the network are missed by more than one $\delta+1$ backbone and this number of missed nodes decreases when the network gets larger but keeping a constant average degree [20]. In summary, LEXICO, which exploits geometric packing properties, offers the best performance in $(\delta+1)$ backbone near domination quality (close to 100% for most backbones) and link length distribution (short virtual links) as well as the strict domatic partition size (DP performance ratio ≥ 0.5 for large graphs). This comes at the price that nodes need to be aware of their locations and a non-localized running time (since the running time increases as n grows for a constant density network). The other coloring algorithms GCOL, LFCOL and 3CL-LAST are comparable and are not better than LEXICO but they provide localized behavior. It is worth noting that the maximum number of disjoint independent dominating sets in a graph G is defined as the independent domatic number $d_i(G)$ which is less than or equal to the domatic number $d(G)$ [13]. In this study, we are effectively finding an experimental approximation of the independent domatic number.

5 Conclusions and Future Work

Our current focus is to perform a more rigorous investigation of the performance guarantees of coloring algorithms in solving the independent domatic partition problem in RGGs. We have shown in this paper that in the context of wireless sensor networks' backbone rotation, graph coloring yields a high quality $(1-\epsilon)$ dominating sets partition with $\epsilon < 0.01$ even for very large graphs. We also intend to integrate the fast $\log^{*} n$ coloring algorithm proposed in [32] in our solution and analyze its performance. Further work involves the study of the spanner properties of our generated backbones and the consideration of wireless communication models other than the ideal unit disk model.

References

1. Alzoubi, K.M., Wan, P., Frieder, O.: Message-optimal connected dominating sets in mobile ad hoc networks. In: Proc. of MOBIHOC 2002, pp. 157–164 (2002)
2. Avin, C.: Fast and efficient restricted delaunay triangulation in random geometric graphs. In: Proc. of Workshop on Combinatorial and Algorithmic Aspects of Networking, CAAN 2005 (2005)
3. Caragiannis, I., Fishkin, A.V., Kaklamanis, C., Papaioannou, E.: A tight bound for online colouring of disk graphs. Theoretical Computer Science 384, 152–160 (2007)
4. Chatterjea, S., Nieberg, T., Zhang, Y., Havinga, P.: Energy-efficient data acquisition using a distributed and self-organizing scheduling algorithm for wireless sensor networks. In: Aspnes, J., Scheideler, C., Arora, A., Madden, S. (eds.) DCOSS 2007. LNCS, vol. 4549, pp. 368–385. Springer, Heidelberg (2007)
5. Couture, M., Barbeau, M., Bose, P., Carmi, P., Kranakis, E.: Location oblivious distributed unit disk graph coloring. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 222–233. Springer, Heidelberg (2007)
6. Diaz, J., Penrose, M.D., Petit, J., Serna, M.J.: Linear orderings of random geometric graphs. In: Widmayer, P., Neyer, G., Eidenbenz, S. (eds.) WG 1999. LNCS, vol. 1665, pp. 291–302. Springer, Heidelberg (1999)
7. Feige, U., Halldorsson, M.M., Kortsarz, G., Srinivasan, A.: Approximating the domatic number. J. of Computing 32(1), 172–195 (2003)
8. Finocchi, I., Panconesi, A., Silvestri, R.: Experimental analysis of simple, distributed vertex coloring algorithms. In: Proc. of SODA 2002, pp. 606–615 (2002)
9. Fotakis, D., Nikoletseas, S., Papadopoulou, V., Spirakis, P.: Hardness results and efficient approximations for frequency assignment problems: Radio labelling and radio coloring. In: Proc. of Workshop on Algorithmic Issues in Communication Networks, vol. 20(2), pp. 121–180 (2001)
10. Funke, S., Milosavljevic, N.: Infrastructure-establishment from scratch in wireless sensor networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 354–367. Springer, Heidelberg (2005)
11. Gavoille, C., Klasing, R., Kosowski, A., Kuszner, L., Navarra, A.: On the complexity of distributed graph coloring with local minimality constraints. Networks 54(1), 12–19 (2009)
12. Gräf, A., Stumpf, M., Weinβenfels, G.: On coloring unit disk graphs. Algorithmica 20, 277–293 (1998)
13. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Fundamentals of Domination in Graphs. CRC Press, Boca Raton (1998)
14. Islam, K., Akl, S.G., Meijer, H.: Distributed generation of a family of connected dominating sets in wireless sensor networks. In: Krishnamachari, B., Suri, S., Heinzelman, W., Mitra, U. (eds.) DCOSS 2009. LNCS, vol. 5516, pp. 343–355. Springer, Heidelberg (2009)
15. Islam, K., Akl, S.G., Meijer, H.: Maximizing the lifetime of a sensor network through domatic partition. In: Proc. of the 34th IEEE Conference on Local Computer Networks (LCN) (2009)
16. Kothapalli, K., Scheideler, C., Onus, M., Richa, A.: Constant density spanners for wireless ad-hoc networks. In: Proc. of the SPAA 2005, pp. 116–125 (2005)
17. Lenzen, C., Suomela, J., Wattenhofer, R.: Local algorithms: Self-stabilization on speed. In: Proc. of 11th International Symposium on Stabilization, Safety and Security of Distributed Systems (SSS), pp. 17–34 (2009)

18. Lin, Z., Wang, D., Xu, L., Gao, J.: A coloring based backbone construction algorithm in wireless ad hoc network. In: Chung, Y.-C., Moreira, J.E. (eds.) GPC 2006. LNCS, vol. 3947, pp. 509–516. Springer, Heidelberg (2006)
19. Mahjoub, D., Matula, D.W.: Experimental study of independent and dominating sets in wireless sensor networks. In: Liu, B., Bestavros, A., Du, D.-Z., Wang, J. (eds.) WASA 2009. LNCS, vol. 5682, pp. 32–42. Springer, Heidelberg (2009)
20. Mahjoub, D., Matula, D.W.: Employing $(1 - \varepsilon)$ dominating set partitions as backbones in wireless sensor networks. In: Proc. of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 98–111 (2010)
21. Marathe, M., Breu, H., Ravi, S., Rosenkrantz, D.: Simple heuristics for unit disk graphs. Networks 25, 59–68 (1995)
22. Matula, D.W., Beck, L.: Smallest-last ordering and clustering and graph coloring algorithms. J. of the ACM 30(3), 417–427 (1983)
23. Matula, D.W., Marble, G., Isaacson, J.: Graph Coloring Algorithms. In: Graph Theory and Computing, pp. 109–122. Academic Press, London (1972)
24. Matula, D.W., Sokal, R.: Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. Geographical Analysis 12, 205–222 (1980)
25. Misra, R., Mandal, C.A.: Efficient clusterhead rotation via domatic partition in self-organizing sensor networks. Wireless Communications and Mobile Computing 9(8), 1040–1058 (2008)
26. Misra, R., Mandal, C.A.: Rotation of cds via connected domatic partition in ad hoc sensor networks. IEEE Trans. Mob. Comput. 8(4), 488–499 (2009)
27. Moscibroda, T., Wattenhofer, R.: Maximizing the lifetime of dominating sets. In: Proc. of 5th IEEE WMAN 2005 (2005)
28. Pandit, S., Pemmaraju, S.V., Varadarajan, K.: Approximation algorithms for domatic partitions of unit disk graphs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 312–325. Springer, Heidelberg (2009)
29. Parthasarathy, S., Gandhi, R.: Distributed algorithms for coloring and domination in wireless ad hoc networks. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 447–459. Springer, Heidelberg (2004)
30. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM, Philadelphia (2000)
31. Pemmaraju, S.V., Pirwani, I.A.: Energy conservation via domatic partitions. In: Proc. of MobiHoc 2006, pp. 143–154 (2006)
32. Schneider, J., Wattenhofer, R.: A log-star distributed maximal independent set algorithm for growth-bounded graphs. In: Proc. of PODC 2008, pp. 35–44 (2008)

On Multihop Broadcast over Adaptively Duty-Cycled Wireless Sensor Networks

Shouwen Lai and Binoy Ravindran

Virginia Tech, ECE Department, Blacksburg, VA 24060, USA
{swlai,binoy}@vt.edu

Abstract. We consider the problem of multihop broadcast over adaptively duty-cycled wireless sensor networks (WSNs) where neighborhood nodes are not simultaneously awake. We present Hybrid-cast, an asynchronous and multihop broadcasting protocol, which can be applied to low duty-cycling or quorum-based duty-cycling schedule where nodes send out a beacon message at the beginning of wakeup slots. Hybrid-cast achieves better tradeoff between broadcast latency and broadcast count compared to previous broadcast solutions. It adopts opportunistic data delivery in order to reduce the broadcast latency. Meanwhile, it reduces redundant transmission via delivery deferring and online forwarder selection. We establish the upper bound of broadcast count and the broadcast latency for a given duty-cycling schedule. We evaluate Hybrid-cast through extensive simulations. The results validate the effectiveness and efficiency of our design.

1 Introduction

Multihop broadcast [17] is an important network service in WSNs, especially for applications such as code update, remote network configuration, route discovery, etc. Although the problem of broadcast has been well studied in always-on networks [12,22] such as wireless ad hoc networks where neighbor connectivity is not a problem, broadcast is more difficult in duty-cycled WSNs where each node stays awake only for a fraction of time slots and neighborhood nodes are not simultaneously awake for receiving data. The problem becomes more difficult in asynchronous [24] and heterogenous duty-cycling [9] scenarios.

To support broadcast, synchronization of wakeup schedules is one promising approach adopted by many duty-cycling MAC protocols, such as S-MAC [23] and T-MAC [4]. Such protocols simplify broadcast communication by letting neighborhood nodes stay awake simultaneously. However, this approach results in high overhead for periodic clock synchronization when compared to the low frequency of broadcast service in WSNs. Since energy is critical to WSNs, energy-efficient asynchronous MAC protocols have become increasingly attractive for data communication, as proposed in B-MAC [14], RI-MAC [18], Disco [5], and quorum-based wakeup scheduling [24,10].

However, previous asynchronous MAC protocols for duty-cycled WSNs mostly focus on unicast communication, and do not work well for broadcasting. One straightforward way to support one-hop broadcast in such cases is to deliver data multiple times for all neighbors, which results in redundant transmissions. With multihop broadcasting to an entire network, the problems are more amplified, as some neighbors attempt to forward the broadcast message while the original transmitting node still attempts to transmit it to other nodes of its neighbors, increasing collisions and wasting energy consumption for transmission.

There have been some efforts in the past to support multihop broadcasting in duty-cycled WSNs. Wang *et al.* [21] transformed the problem into a shortest-path problem with the assumption of duty-cycle awareness, which is not valid for asynchronously duty-cycled WSNs. DIP [16], ADB [17], and opportunistic flooding [6] were designed with a smart gossiping approach. Essentially, these protocols use unicast to replace broadcast for flooding, toward reducing the flooding latency in the entire network. However, they may lack efficiency in large-scale networks or on delivering large chunks of data to entire network because message cost and higher transmission energy consumption.

To overcome the disadvantages of replacement via pure unicast, we present Hybrid-cast, an asynchronous broadcast protocol for broadcasting with low latency and reduced message count. In Hybrid-cast, a node only forwards a message to neighbors who wake up and send out beacon messages. A node defers broadcasting by one or more time slot(s) after receiving the beacon message from the first awake neighbor in order to wait for more nodes that may potentially wake up, so that more nodes are accommodated in one broadcast. It also adopts online forwarder selection in order to reduce the transmission redundancy. Compared with previous protocols, Hybrid-cast can achieve less broadcast latency and smaller message count.

The rest of the paper is organized as follows: We discuss related works in Section 2. In Section 3, we state our models, assumptions, and preliminaries. In Section 4, we present the design of Hybrid-cast. We theoretically analyze the performance of Hybrid-cast in Section 5, and provide further discussions in Section 6. Simulation results are presented in Section 7. We conclude in Section 8.

2 Past and Related Works

We review past and related efforts on broadcast solutions for duty-cycled WSNs. Due to space constraints, we omit reviews for always-on multihop networks.

Gossip or opportunistic approach. Opportunistic unicast routing, like EXOR [1], was proposed to exploit wireless broadcast medium and multiple opportunistic paths for efficient message delivery. Regarding broadcasting, the main purpose of opportunistic approach aimed at ameliorating message implosion. Smart Gossip [8] adaptively determines the forwarding probability for received flooding messages at individual sensor nodes based on previous knowledge and network topology.

In Opportunistic Flooding [6] (abbreviated as OppFlooding), each node makes probabilistic forwarding decisions based on the delay distribution of next-hop nodes. Only opportunistic early packets are forwarded via the links outside of the energy-optimal tree to reduce flooding delays and the level of redundancy. To resolve decision conflicts, the authors build a reduced flooding sender set to alleviate the hidden terminal problem. Within the same sender set, the solution uses a link-quality-based backoff method to resolve and prioritize simultaneous forwarding operations. The main problem of pure opportunistic flooding is the overhead in terms of transmission times.

Synchronized or duty-cycle awareness. Wang et al. [21] present a centralized algorithm, mathematically modeling the multihop broadcast problem as a shortest-path problem in a time-coverage graph, and also present two similar distributed algorithms. However, their work simplifies many aspects necessary for a complete MAC protocol, and may not be appropriate for real implementation. The work also assumes duty-cycle awareness, which makes it difficult to use it in asynchronous WSNs since duty-cycle awareness needs periodic time-synchronization due to clock drifting. RBS [20] proposes a broadcast service for duty-cycled sensor networks and shows its effectiveness in reducing broadcast count and energy costs.

All these works based on synchronization assume that there are usually multiple neighbors available at the same time to receive the multicast/flooding message sent by a sender. This is not true in low duty-cycled asynchronous networks.

Asynchronous solution. B-MAC [14] can support single-hop broadcast in the same way as it supports unicast, since the preamble transmission over an entire sleep period gives all of the transmitting node's neighbors a chance to detect the preamble and remain awake for the data packet. X-MAC [3] substantially improves B-MAC's performance for unicast, but broadcast support is not clearly discussed in that paper. X-MAC is not promising for broadcast since the transmitter has to continually trigger the neighbors to wake up.

ADB [17] avoids the problems faced by B-MAC and X-MAC by efficiently delivering information on the progress of each broadcast. It allows a node to go to sleep immediately when no more neighbors need to be reached. ADB is designed to be integrated with an unicast MAC that does not occupy the medium for a long time, in order to minimize latency before forwarding a broadcast. The effort in delivering a broadcast packet to a neighbor is adjusted based on link quality, rather than transmitting throughout a duty cycle or waiting throughout a duty cycle for neighbors to wake up. Basically, ADB belongs to the unicast replacement approach and it needs significant modification to existing MAC protocols for supporting broadcast.

3 Models and Preliminaries

3.1 Network Model and Assumptions

We model a multi-hop wireless sensor network as a directed graph $G(V, E)$, where V is the set of nodes, and E is the set of edges. If node v_j is within the transmission range of node v_i, then an edge (v_i, v_j) is in E. We assume bidirectional links. We use the term "connectivity" loosely in our model, in the sense that a topologically connected network in our context may not be connected at any time; instead, all nodes are reachable from a node within a finite amount of time by the underlying MAC protocol. We define the one-hop neighborhood of node n_i as $N(i)$.

We assume that time axes are arranged as consecutive short time slots, all slots have the same duration T_s, and each node n_i adopts a periodic wakeup schedule every L_i time slots. The wakeup schedule can be once every L_i slots or based on quorum schedules (i.e., cyclic quorum systems or grid quorum systems [11]). L_i is called cycle length for node n_i. We assume that beacon messages are sent out at the beginning of wakeup slots, as in [18,10]. When a node wants to transmit messages, it will wait until beacons are received from neighbors.

We also make the following assumptions: (1) There is no time synchronization between nodes (thus the time slots in two nodes are not necessarily aligned); (2) The overhead of turning on and shutting down radio is negligibly small compared with the long duration of time slots (i.e., $50ms \sim 500ms$); (3) There is only one sink node in the network (but our solution can be easily extend to the scenario of multiple sink nodes).

3.2 Heterogenous Wakeup Scheduling

Heterogenous wakeup scheduling means that nodes adopt different wakeup schedules independently to reflect their remaining energy. How to configure this schedule (i.e., the value of L_i) has been described by past works such as [19], and is outside the scope of our work.

We consider two types of heterogenous wakeup scheduling approaches: low duty-cycling schedule and quorum duty-cycling schedule. Low duty-cycling means that a node wakes up one slot for every n_i (n_i is an integer) time slots. For example, in Figure 1(a), receiver 1 has a schedule of $[1, 0, 0]$, where 1 means wakeup slot, and receiver 2 has the schedule of $[1, 0, 0, 0]$. They do not always overlap on wakeup slots.

For quorum-based duty cycling, wakeup scheduling follows a quorum system [11] design. In quorum-based duty cycling, two neighbor nodes can hear each other at least once within limited time slots via the non-empty intersection property of quorums. We choose cyclic quorum system [10] in this paper. But our work can also be applied to other quorum systems.

We use the following definitions for briefly reviewing quorum systems (which are used for wakeup scheduling).

Let n denote a cycle length and $U = \{0, \cdots, n-1\}$.

Definition 1. *A* quorum system $\mathcal{Q}$ *under* U *is a superset of non-empty subsets of* U*, each called a quorum, which satisfies the intersection property:* $\forall G, H \in \mathcal{Q} : G \cap H \neq \emptyset$*. If* $\forall G, H \in \mathcal{Q}$*,* $i \in \{0, 1, ...n-1\}$*:* $G \cap (H + i) \neq \emptyset$*, where* $H + i = \{(x + i) \bmod n : x \in H\}$*.* $\mathcal{Q}$ *is said to have the* rotation closure property

A cyclic quorum system (cqs) satisfies the rotation closure property, and is denoted as $C(A, n)$ where A is a quorum and n is the cycle length. For example, the cqs $\{\{1, 2, 4\}, \{2, 3, 5\} \cdots, \{7, 1, 3\}\}$ can be denoted as $C(\{1, 2, 4\}, 7)$. The wakeup schedule complying with $C(\{1,2,4\},7)$ are $[1,1,0,1,0,0,0]$ and its rotations as shown in Figure 1(b).

For two different cyclic quorum systems $C(A_1,n_1)$ and $C(A_2,n_2)$, if two quorums from them, respectively, have non-empty intersections even with drifting clocks, they can be used for heterogenous wakeup scheduling in WSNs as proved by in [10]. For example, given $C(\{1,2,4\},7)$ and $C(\{1,2,4,10\},13)$, two quorums from them, respectively, will have non-empty intersection for every 13 time slots. Therefore, two nodes that wake up with schedules complying with any two quorums from the two cyclic quorum systems can hear each other.

3.3 Problem Statement

Let us define the broadcast latency as the time between the beginning of a broadcast and the time at which every node receives the broadcast message. Also, let us define the broadcast count as the number of broadcasting via all nodes to ensure that the entire network receives the message. Our goal is to design a broadcast schedule, which can not only shorten the broadcast latency but also the broadcast count for flooding a message to the entire network. The protocol that we present, Hybrid-cast, is a heuristic solution to this problem.

4 The Hybrid-Cast Protocol

4.1 Overview

In Hybrid-cast, a transmitter will stay awake for long enough time to hear the beacon message from its neighbors. Due to heterogenous wake-up scheduling, for low duty-cycling, the node will stay awake for L_m time slots, which is the largest cycle length of all neighbors. By doing this, it can hear beacons from all neighbors. For quorum duty-cycling, the transmitter will switch to the wakeup schedules which has the largest cycle length from all its neighbors.

Hybrid-cast adopts opportunistic forwarding with delivery deferring to shorten broadcast latency and broadcast count: the transmitter will forward the message

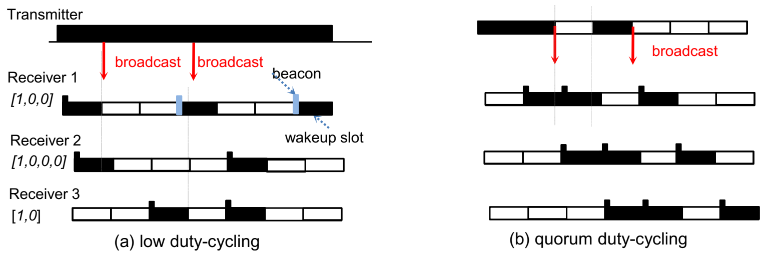


Fig. 1. Opportunistic broadcasting with delivery deferring (a) low duty-cycling case; (b) quorum duty-cycling case with wakeup schedules of [1,1,0,1,0,0,0] and its rotations which comply with (7,3,1) cqs design in [10]

within δ time after it hears the beacon messages from early-wakeup neighbors, rather than forwarding immediately after hearing the beacon messages. An illustration is given in Figure 1(a). Here, δ (i.e., $\delta = T_s$ for low duty-cycling) is called the deferring time. By deferring, the first-awake neighbor can still receive the broadcast message. Meanwhile, more neighbors which wake up during the deferred time period can receive the broadcast message, so that less number of broadcast is necessary for one-hop broadcasting.

To further reduce redundant transmissions, Hybrid-cast adopts online forwarder selection. "Online" means that a node selects the least relay node among its instant one-hop awake neighbors, rather than all one-hop neighbors, to cover its two hop neighbors, in order to reduce transmission redundancy and collision.

4.2 Wakeup Schedule Switching

Due to adaptive duty-cycling, neighbor discovery becomes more difficult. In order to hear the beacon message from all neighbors, a node must switch its wakeup schedule for staying awake for enough time slots.

For the case of low duty-cycling, in the idle state, a node n_i follows its own wakeup schedule. If the node needs to forward a broadcast message (i.e., the node is selected as a relay node), n_i should stay awake for at least L_m slots, where $L_m = \max_{n_j \in N(i)}\{L_j\}$. By doing this, n_i can hear beacon messages from all neighbors within the minimum necessary time slots.

For the case of quorum duty-cycling, n_i just switches to the schedule of the node which has the longest cycle length. Due to the non-empty intersection property [10], n_i can still hear all neighbors even when it does not stay awake in every time slot of a whole cycle length.

A node needs to know the largest cycle length of its neighbors before schedule switching. This can be achieved by either pre-setting the largest global cycle length or by dynamic neighbor information exchange protocols.

4.3 Opportunistic Forwarding with Deferring

Opportunistic forwarding means that a transmitter forwards data immediately to the neighbor which wake up earlier, for minimizing broadcast latency. Previous efforts on opportunistic flooding such as [6] use unicast for broadcasting. However, opportunistic forwarding via pure unicast suffers from large broadcast count.

In Hybrid-cast, broadcast deferring is adopted to minimize the one-hop broadcast count. By deferring, a transmitter will not broadcast messages immediately after receiving the beacon from the first-awake neighbor. In order to ensure that more neighbors receive the broadcast message, the transmitter defers the broadcasting by $\delta = 1$ *time slot*. By doing this, the first-awake neighbor can still receive the message, and the neighbors which wake up before the deferring time is due can also receive the broadcast message. Thus, deferring combines the advantages of opportunistic forwarding and the advantages of broadcasting over wireless radio.

As shown in Figure 1, suppose there are three neighbors for the transmitter. The transmitter only needs to broadcast two times (marked by the red arrow) to ensure that all neighbors will receive the message. This is more efficient than the pure opportunistic forwarding mechanism.

The only disadvantages of deferring is the additional latency (1 time slot for one-hop broadcasting) for flooding to the entire network. Therefore, deferring allows the tradeoff between the number of broadcast count and the broadcast latency to be exploited. We show in Section 5.2 that such additional latency is relatively small for the low duty-cycling case.

4.4 Online Forwarder Selection

In order to reduce the broadcast count or redundant transmission for multihop broadcasting, it is necessary to select as small number of relay nodes as possible. Many past efforts have formulated this problem as the Minimum Connecting Dominating Set (MCDS) problem [2]. However, we argue that a static MCDS cannot be applied for relay node selection in Hybrid-cast. First, to shorten the latency, it is necessary to select the relay nodes or forwarders along the direction of opportunistic forwarding, which results in online (or live) forwarder selection, rather than a static topology control as done in MCDS. Secondly, MCDS does not achieve minimum broadcast count in asynchronous duty-cycled WSNs due to multiple delivery for single hop broadcasting.

Algorithm 1. Algorithm for all node n_x:

```
1: set $N_{awake}(x)$;
2: $N^2_{reachable}(x) = \cup_{y \in N_{awake}(x)} N(y) - N_{awake}(x)$;
3: for $n_y \in N_{awake}(x)$ do
4:     if node $n_u \in N^2_{reachable}(x)$ which is only reachable by $n_y$ then
5:         $n_y$ is selected into O-$MPR(x)$;
6:         $N^2_{reachable}(x) = N^2_{reachable}(x) - n_u$;
7: while $N^2_{reachable}(x) \neq \emptyset$ do
8:     for $n_u \in N_{awake}(x)$ - O-$MPR(x)$ do
9:         $n_m = n_u$ which covers the most nodes in $N^2_{reachable}(x)$;
10:    $n_m$ is selected into O-$MPR(x)$;
11:    $N^2_{reachable}(x) = N^2_{reachable}(x)$ - node set covered by $n_m$;
```

In Hybrid-cast, initially, each node maintains its one hop awake neighbors (defined as $N(x)$) and the set of two hop neighbors $N^2(x)$ based on any underlying neighbor discovery protocols. The sink node or a relay node n_x computes the least number of relay nodes among its one-hop awake neighbors (defined as $N_{awake}(x)$) to cover the reachable two hop neighbors (defined as $N^2_{reachable}(x)$).

$$N^2_{reachable}(x) = \cup_{y \in N_{awake}(x)} N(y) - N_{awake}(x) \qquad (1)$$

The main purpose of the online forwarder selection algorithm in a transmitter n_x is to compute $N^2_{reachable}(x)$ as shown in Equation 1, and to compute the minimum number of relays to cover $N^2_{reachable}(x)$.

We adopt a heuristic solution, which is similar to the minimum multipoint relays (MPR) algorithm in [15]. The MPR problem is NP-Complete as shown in [15]. Thus, the minimum online forwarder selection problem is also NP-Complete. We denote the online MPR set for the transmitter n_x as O-$MPR(x)$. We provide a heuristic algorithm for computing O-$MPR(x)$ as described in Algorithm 1. An illustration is given in Figure 2(a).

Let us define the delivery latency from node n_i to node n_j as the time between when the data is ready in n_i and time at which the broadcast data is received by the neighbors, and denote the latency as $\tau_{i,j}(t)$ at time t ($\tau_{i,j}(t)$ is varying at different time). We have the following property.

Theorem 1. *Suppose node n_i has two neighbor n_j and n_k which are one hop away from each other. Then, at a time instant, t_i, we have the triangular property:*

$$\tau_{i,j}(t_i) \le \tau_{i,k}(t_i) + \tau_{k,j}(t_i + \tau_{i,k}(t_i)) \tag{2}$$

Proof. Suppose at time t_i, the data arriving time slot at n_j is t_j, and the data arriving time at n_k is t_k.

If $t_k \le t_j$, which means that the data arriving time at n_k is earlier than the data arriving time at n_j, $\tau_{i,k}(t_i + \tau_{k,j}(t_i + \tau_{i,k}(t_i)) = t_k - t_i + t_j - t_k = t_j - t_i = \tau_{i,j}(t_i)$. Otherwise, if $t_k > t_j$, which means that the data arriving time at n_k is later than the data arriving time at n_j, we have $\tau_{i,k}(t_i) + \tau_{k,j}(t_i + \tau_{i,k}(t_i)) = t_k - t_i + t_j^{'} - t_k > t_j - t_i + t_j^{'} - t_k > t_j - t_i > \tau_{i,j}(t_i)$. The theorem follows.

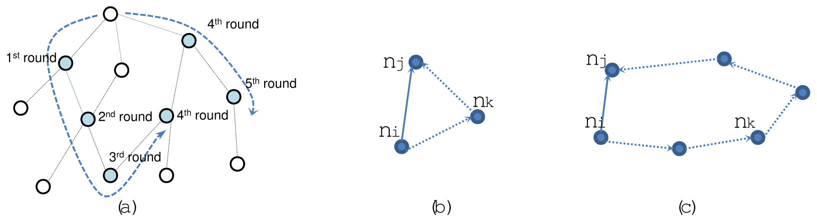


Fig. 2. Online forwarder selection and the triangular path condition

Theorem 1, as illustrated in Figure 2(b), illustrates that node n_i will always broadcast data to its one-hop neighbor n_j directly, without through other nodes.

We also have the following property.

Lemma 1. Triangular Path Condition*: For a node n_i and its neighbor n_j, at any time, the one-hop broadcast latency $n_i \rightarrow n_j$ is always the minimum possible.*

We omit the proof for the triangular path condition since it is a simple extension from that of Theorem 2. An illustration is given in Figure 2(c). Note that the triangular path condition does not exist in static networks. The triangular path condition indicates that the one-hop direct broadcast always achieves the least latency, in adaptively duty-cycled WSNs.

5 Performance Analysis

We now analyze the performance of Hybrid-cast in terms of the broadcast count and the broadcast latency, in order to illustrate its design advantages.

5.1 Upper-Bound on One-Hop Broadcast Count

We consider two scenarios in analyzing the one-hop broadcast count. In the low duty-cycling scenario, the schedule for a node n_i is waking up once every L_i time slots. In the quorum duty-cycling scenario, the schedule for a node n_i is waking up q times for every L_i consecutive time slots, where q is the quorum size.

Lemma 2. *[low duty-cycling] In Hybrid-cast, for a node n_i, the broadcast count is at least one, and at most* $\max\{\Delta, L_m\}$*, where Δ is the node degree of n_i and L_m is the maximum cycle length of nodes in the neighborhood.*

Proof. If all nodes wake up within the same time slot, then after broadcast deferring, the transmitter can hear all neighbors, and one broadcast can cover all neighbors.

Otherwise, if $\Delta \geq L_m$, the transmitter can hear all neighbors via staying awake for L_m time slots. Therefore, the maximum broadcast count is L_m. If $\Delta < L_m$, the transmitter can hear neighbors for at most Δ times, and the maximum broadcast count is Δ. Thus, the maximum number of broadcast count is $\max\{\Delta, L_m\}$.

By the Lemma 2, the upper bound of broadcast count in Hybrid-cast is at most n (where n is the network size) in the ideal case.

Lemma 3. *[quorum duty-cycling] In Hybrid-cast, for node n_i, the broadcast count is at least one, and at most* $\max\{\Delta, q_m\}$*, where Δ is the node degree of n_i and q_m is the largest quorum size of the quorum systems adopted by nodes in the neighborhood.*

Proof. If all nodes wake up within one time slot, then after broadcast deferring, the transmitter can hear all neighbors, and one broadcast can cover all neighbors.

Otherwise, if $\Delta \geq q_m$, the transmitter can hear all neighbors via staying awake in time slots scheduled by the quorum design. Therefore the maximum broadcast count is q_m. If $\Delta < q$, the transmitter will hear neighbors for at most Δ times, and the maximum broadcast count is Δ. Thus, the maximum broadcast count is $\max\{\Delta, q_m\}$.

5.2 Delivery Latency

Lemma 4. *Suppose the depth of the network (i.e., maximum layers by breadth-first-search) is D_{max}. Then, the upper bound for delivery latency is $L_m * D_{max} * T_s$ in low duty-cycling mode, where L_m is the maximum cycle length of nodes in the network. The upper bound is $q_m * D_{max} * T_s$ for quorum duty-cycling mode, where q_m is the largest quorum size of the quorum systems adopted by all nodes in the network.*

Proof. Based on the Triangular Path Condition in Lemma 1, a node always broadcasts a message to its one-hop neighbors directly. Thus, for one hop broadcasting, the latency is at most $L_m * T_s$. After $L_m * T_s$, all nodes in the first layer will receive the broadcast message. Therefore, after $L_m * D_{max} * T_s$ time, all nodes in the network will receive the broadcast message.

6 Discussion

Note that we do not assume local synchronization or duty-cycle awareness, which is required by past works such as [6] and [21]. The assumption in Hybrid-cast is neighbor-awareness. Such awareness can be achieved by neighbor discovery protocols, or by quorum-based duty-cycling [10]. Each node will inform its neighbors after reconfiguration on the duty-cycling.

By adopting quorum duty-cycling, Hybrid-cast can be extended to mobile WSNs, because neighbor discovery is guaranteed within bounded time in quorum duty-cycling, as shown in [10].

Due to the problem of hidden terminal, it is possible that one node may receive broadcast messages from two nodes simultaneously, which leads to collision. For reliable broadcasting, if a node received the broadcast, it can set a mark field in the beacon message. By checking the beacon message from the neighbor, a transmitter can decide whether retransmission is necessary. We do not defer broadcast for retransmission. The transmitter could backoff a random period $0 \leq t \leq T_s$ in order to avoid collision.

We do not explicitly consider reliability issues in Hybrid-cast. However, the traditional ACK and NACK mechanisms for reliable data transmission can be applied to Hybrid-cast to support reliable broadcasting.

7 Simulation Results

We simulated Hybrid-cast using the OMNET++ simulator [13] and compared it against ADB [17] and opportunistic broadcasting [6] (denoted as OppFlooding).

Our experimental settings were consistent with the configurations in [6,7]. We set the wireless loss rate as 0.1 and the duration of one time slot as 100 ms. The wireless communication range was set to 10m. We adopted the wireless loss model in [25], which considers the oscillation of radio. The size of the broadcast message packets was fixed as 512 bytes.

We examined the two main factors that affect the performance of our algorithms, including network size and duty-cycle setting. We generated a network with different number of nodes. For each network size, we randomly generated 10 topologies. Each data point reported in this section is the average of 10 topologies, with 10 runs on each topology. We varied the network size to understand its impact on the broadcast count and broadcast latency.

We measured the performance of the algorithms in a variety of duty cycle settings. For the low duty-cycling scenario, we varied the duration of the total periodic cycle length from $2T_s$ to $10T_s$ to generate heterogenous duty-cycling in a network for different nodes. For the quorum duty-cycling case, we choose the $(7, 3, 1)$, $(13, 4, 1)$, and $(21, 5, 1)$ difference sets for the heterogenous schedule settings. Since ADB, OppFlooding, and Hybrid-cast are independent of wakeup scheuling, we argue that the comparison is fair, even though ADB and OppFlooding do not explicitly support quorum duty-cycling.

7.1 Broadcast Count

We first measure the broadcast count which is the total number of broadcasting for flooding a message to the whole network. In this set of experiments, the network size was fixed by 200 nodes. The experimental results for different protocols are shown in Figure 3(a). In the low duty-cycling case, Hybrid-cast outperforms ADB by approximately 50%, because of the less number of unicasts involved, due to the protocol's deferring and online forwarder selection.

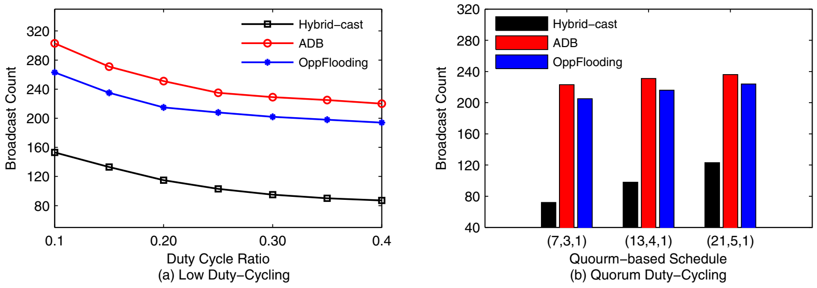


Fig. 3. Performance comparison on broadcast count

For the quorum-based duty cycle setting, all nodes in the network chose homogenous quorum schedules. The setting was varied simultaneously for all nodes in different set of experiments. As shown in Figure 3(b), Hybrid-cast performs better since broadcasting are aggregated within quorum slots in each cycle. For example, for the $(7, 3, 1)$ setting (i.e., a node will stay awake at the 1^{st}, 2^{nd}, and 4^{th} slot on every 7 consecutive slots), there are at most 3 broadcasts to ensure that all neighbor nodes receive the broadcast message. However, for ADB and OppFlooding, the average one-hop broadcast count was 5 or 6, given the average degree in the network that we configured. The results validate the performance analysis in Section 5.1.

7.2 Broadcast Latency

Figure 4(a) shows the broadcast latency (defined as the time from broadcast beginning to all nodes receiving the broadcast data). With deferring, Hybrid-cast has slightly higher latency than ADB and OppFlooding, by about 10%, when the duty cycle ratio is 0.4, and by about 5%, when the duty cycle ratio is 0.1. As shown in Figure 4(a), as the duty cycle ratio decreases, the disadvantages of Hybrid-cast become more negligible, since the broadcast latency is more dominated by neighbor discovery latency.

For the case of quorum duty-cycling, as shown in Figure 4(b), we observe a similar trend as that of low duty-cycling. The latencies for all three protocols tend to increase with larger quorum cycle. However, the latencies tend to converge to the same value when the quorum cycle increases. This is because, the neighbor

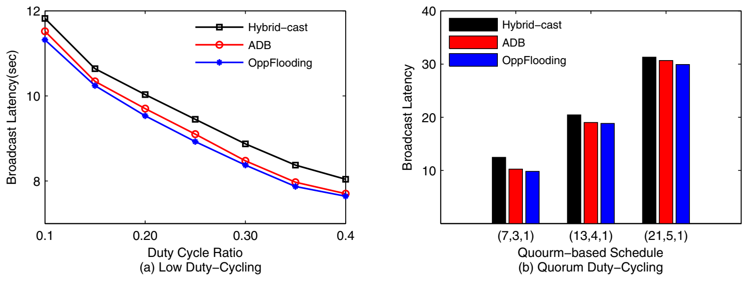


Fig. 4. Performance comparison on broadcast latency

discovery latency is approximately linearly increasing with quorum cycle, as shown in [10] The results also validate the performance analysis in Section 5.2.

7.3 Impact of Network Size

We also evaluated the impact of network size and heterogenous duty-cycling on message count and broadcast latency. For the low duty-cycling case, each node randomly selected a duty cycle ratio in the range 0.1 to 0.4. For the quorum duty-cycling case, we chose the $(7, 3, 1)$, $(13, 4, 1)$, and $(21, 5, 1)$ difference sets for the schedules of all nodes (the non-empty intersection property among these sets was proved in [10]). In the simulation experiments, we varied the network size from 200 nodes to 1600 nodes.

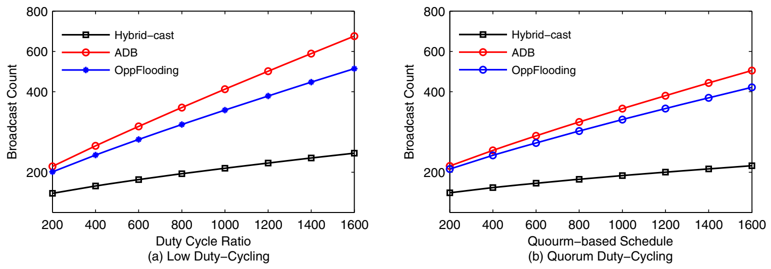


Fig. 5. Broadcast count with different network size

As shown in Figure 5, as the network size increases, the message count of Hybrid-cast and the other two solutions exhibit an increasing trend. This is because, more relay nodes will be selected in larger networks. The same trend exists for broadcast latency as shown in Figure 6, as there are more hops along the breadth-first-search tree. This is consistent with the analysis in Section 5.2.

We also evaluated the impact of network size for quorum-based duty cycle setting. We observed similar trends for the broadcast count and broadcast latency as that in the low duty-cycling setting. The performance comparisons thus illustrate the performance tradeoff achieved by Hybrid-cast.

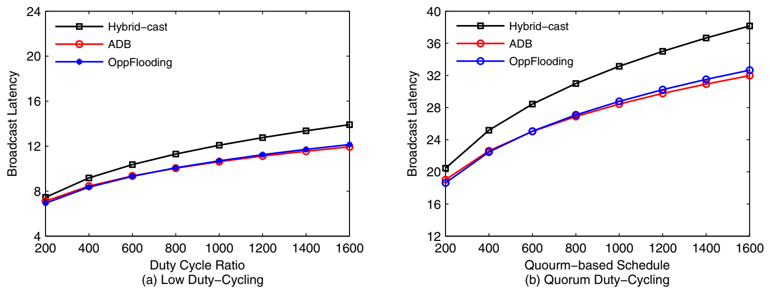


Fig. 6. Broadcast latency with different network size

8 Conclusions

In this paper, we designed an asynchronous broadcasting protocol, Hybrid-cast, for WSNs with adaptively low duty-cycling or quorum-based duty-cycling schedules. The main difficulty of this problem is that, sensor nodes are not time-synchronized and do not stay awake simultaneously. Hybrid-cast broadcasts messages to the neighbors who wake up early, in order to shorten the broadcast latency. Previous solutions often use multiple unicasts for broadcasting, which incurs high overhead. To overcome the disadvantages of such multiple unicasts, Hybrid-cast defers broadcasting to ensure that the number of awake neighbors is as large as possible. We also selected the minimum relay points online in order to reduce broadcast count and collisions.

We mathematically established the upper bound of broadcast count and broadcast latency for a given duty-cycling schedule. We compared the performance of Hybrid-cast with ADB and OppFlooding protocols. Our simulation results validated the effectiveness and efficiency of our design.

References

1. Biswas, S., Morris, R.: Exor: opportunistic multi-hop routing for wireless networks. SIGCOMM Comput. Commun. Rev. 35(4), 133–144 (2005)
2. Blum, J., Ding, M., Thaeler, A., Cheng, X.: Connected dominating set in sensor networks and manets. In: Handbook of Combinatorial Optimization, pp. 329–369 (2005)
3. Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In: ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 307–320 (2006)
4. Dam, T.V., Langendoen, K.: An adaptive energy-efficient mac protocol for wireless sensor networks. In: ACM SenSys (2003)
5. Dutta, P., Culler, D.: Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In: ACM SenSys, pp. 71–84 (2008)
6. Guo, S., Gu, Y., Jiang, B., He, T.: Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. In: ACM Conference on Mobile Computing and Networking (MobiCom), pp. 133–144 (2009)
7. Jurdak, R., Baldi, P., Lopes, C.V.: Adaptive low power listening for wireless sensor networks. IEEE Transactions on Mobile Computing 6(8), 988–1004 (2007)

8. Kyasanur, P., Choudhury, R.R., Gupta, I.: Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 91–100 (October 2006)
9. Lai, S., Ravindran, B.: On distributed time-dependent shortest paths over duty-cycled wireless sensor networks. In: IEEE International Conference on Computer Communications (INFOCOM) (2010)
10. Lai, S., Zhang, B., Ravindran, B., Cho, H.: Cqs-pair: Cyclic quorum system pair for wakeup scheduling in wireless sensor networks. In: Baker, T.P., Bui, A., Tixeuil, S. (eds.) OPODIS 2008. LNCS, vol. 5401, pp. 295–310. Springer, Heidelberg (2008)
11. Luk, W.S., Huang, T.T.: Two new quorum based algorithms for distributed mutual exclusion. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 100–106 (1997)
12. Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., Sheu, J.-P.: The broadcast storm problem in a mobile ad hoc network. In: 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pp. 151–162 (1999)
13. OMNET++, http://www.omnetpp.org/
14. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys, pp. 95–107 (2004)
15. Qayyum, A., Viennot, L., Laouiti, A.: Multipoint relaying for flooding broadcast messages in mobile wireless networks. In: 35th Annual Hawaii International Conference on System Science, pp. 3866–3875 (2002)
16. Stann, F., Heidemann, J., Shroff, R., Murtaza, M.Z.: Rbp: robust broadcast propagation in wireless networks. In: ACM SenSys, pp. 85–98 (2006)
17. Sun, Y., Gurewitz, O., Du, S., Tang, L., Johnson, D.B.: Adb: an efficient multihop broadcast protocol based on asynchronous duty-cycling in wireless sensor networks. In: ACM SenSys, pp. 43–56 (2009)
18. Sun, Y., Gurewitz, O., Johnson, D.B.: Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In: ACM Sensys, pp. 1–14 (2008)
19. Vigorito, C.M., Ganesan, D., Barto, A.G.: Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In: IEEE SECON 2007, pp. 21–30 (June 2007)
20. Wang, F., Liu, J.: Rbs: A reliable broadcast service for large-scale low duty-cycled wireless sensor networks. In: IEEE International Conference on Communications (ICC), pp. 2416–2420 (May 2008)
21. Wang, F., Liu, J.: Duty-cycle-aware broadcast in wireless sensor networks. In: IEEE International Conference on Computer Communications (INFOCOM), pp. 468–476 (2009)
22. Williams, B., Camp, T.: Comparison of broadcasting techniques for mobile ad hoc networks. In: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc), pp. 194–205 (2002)
23. Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Transactions on Networking 12, 493–506 (2004)
24. Zheng, R., Hou, J.C., Sha, L.: Asynchronous wakeup for ad hoc networks. In: MobiHoc, pp. 35–45 (2003)
25. Zuniga, M., Krishnamachari, B.: Analyzing the transitional region in low power wireless links. In: IEEE SECON, pp. 517–526 (2004)

A Novel Mobility Management Scheme for Target Tracking in Cluster-Based Sensor Networks[⋆]

Zhibo Wang[1], Wei Lou[2], Zhi Wang[1], Junchao Ma[2], and Honglong Chen[2]

[1] State Key Laboratory of Industrial Control Technology,
Zhejiang University, Hangzhou, P.R. China
{zbwang,wangzhi}@iipc.zju.edu.cn
[2] Department of Computing,
The Hong Kong Polytechnic University, Hong Kong
{csweilou,csjma,cshlchen}@comp.polyu.edu.hk

Abstract. Target tracking is a typical and important application of wireless sensor networks (WSNs). In the consideration of scalability and energy efficiency for target tracking in large scale WSNs, it has been employed as an effective solution by organizing the WSNs into clusters. However, tracking a moving target in cluster-based WSNs suffers the boundary problem when the target moves across or along the boundary among clusters. In this paper, we propose a novel scheme, called *hybrid cluster-based target tracking* (HCTT), which integrates on-demand dynamic clustering into a cluster-based WSN for target tracking. To overcome the boundary problem, when the target moves close to the boundary among clusters, a dynamic cluster will be constructed for the management of target tracking. As the target moves, static clusters and on-demand dynamic clusters alternately manage the tracking task. Simulation results show that the proposed scheme performs better in tracking the moving target when compared with other typical target tracking protocols.

Keywords: Wireless sensor networks; target tracking; mobility management; boundary problem.

1 Introduction

Target tracking is considered important in WSNs as it is a base for many practical applications, such as battlefield surveillance, emergency rescue, disaster response and patient monitoring [1]. Generally speaking, target tracking aims to detect the presence of a target and compute reliable estimates of the locations while the target moves within the area of interest, and forward these estimates to the base station in a timely manner.

It is known that the cluster structure can provide benefits for large scale WSNs. For example, it facilitates spatial reuse of resources to increase the system capacity [2]; it also benefits local collaboration and routing [3] [4]. Recently, cluster structure is gradually adopted for solving the target tracking problem [5, 6, 7]. [5, 6] use dynamic clustering approaches that dynamically wake up a group of nodes to construct a cluster

[⋆] This work is supported in part by grants NSFC No. 60873223, NSFC No. 90818010, ZJU-SKL ICT0903, PolyU 5243/08E, PolyU 5253/09E, and 1-ZV5N.

for local collaboration when the target moves into a region. As dynamic clustering only actives the most appropriate nodes in a cluster at one time while keeping other nodes in the sleep state, it is an energy-efficient way for target tracking. However, dynamic clustering has several drawbacks. First, the dynamic clustering process, repeating as the target moves, incurs much overhead for constructing and dismissing clusters. Second, dynamic clustering mainly considers the aspect of energy-efficient local collaboration without considering efficient data routing to the sink which is another important aspect of target tracking. In contrast, [7] is based on the static cluster structure for the scalability and energy efficiency of target tracking. It uses a predictive mechanism to inform cluster heads about the approaching target, and then the corresponding cluster head wakes up the most appropriate nodes right before the arrival of the target. Compared with dynamic clustering, this approach takes advantages of the cluster-based structure. It addresses the issue of a scalable architecture for coordinating a WSN for target tracking, and data can be easily routed to the sink with low time delay. Besides, the overhead is saved as the tracking task is handed over from one static cluster to another without the dynamic clustering process. The drawback is that the static cluster membership prevents sensors in different clusters from sharing information, which causes a *boundary problem* when the target moves across or along the boundary among clusters. The boundary problem will result in the increase of tracking uncertainty or even the loss of the target. Therefore, tradeoffs should be taken between the energy efficiency and local collaboration.

In this paper, we propose a novel fully distributed mobility management scheme, called *hybrid cluster-based target tracking* (HCTT), for efficient target tracking in large-scale cluster-based WSNs. As shown in Fig.1, when the target is inside a cluster, the static cluster is responsible for target tracking; as the target moves close to the boundary among clusters, an on-demand dynamic clustering process will be triggered to manage the tracking task to avoid the boundary problem. The on-demand dynamic cluster will dismiss soon after the target moves away from the boundary. As shown in Fig.1, the consecutive clusters for tracking the target are $A \rightarrow D1 \rightarrow C \rightarrow D2 \rightarrow E$. As the target moves in the network, static clusters and on-demand dynamic clusters alternately

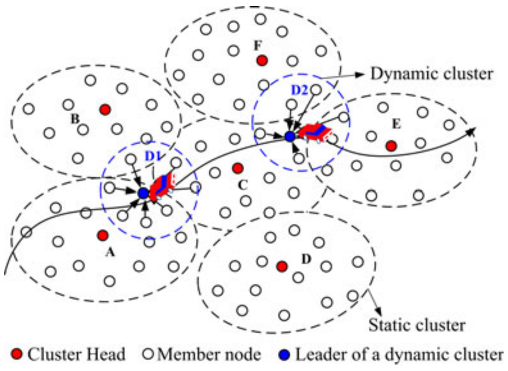


Fig. 1. Illustration of HCTT for target tracking in cluster-based WSNs

manage the tracking task. By integrating on-demand dynamic clustering into a scalable cluster-based structure, the energy efficiency and local sensor collaboration are well balanced.

The main contributions of this paper are summarized as follows: (1) We address the boundary problem for target tracking in cluster-based WSNs. (2) We present a novel hybrid cluster-based target tracking scheme, which balances well between the energy efficiency and local sensor collaboration, to solve the boundary problem in cluster-based networks. The proposed scheme is fully distributed with no help of global information or prediction algorithms. (3) We conduct simulations to show the efficiency of the proposed scheme compared with other typical target tracking solutions.

2 Related Work

The problem of target tracking with WSNs has received considerable attention from various angles and a lot of protocols have been proposed [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. Zhao et al. propose an information-driven dynamic sensor collaboration mechanism for target tracking [8]. Brooks et al. present a distributed entity-tracking framework for sensor networks [9]. Vigilnet, an energy-efficient and real time integrated system for target tracking is designed and implemented in [10, 11]. Recently, Bhatti and Xu present a survey of target tracking using WSNs [12]. In general, target tracking protocols can be mainly classified into three categories: tree-based, cluster-based and prediction-based tracking protocols.

Tree-based tracking protocols: Zhang and Cao propose a dynamic convoy tree-based collaboration (DCTC) framework to detect and track the mobile target [14]. DCTC relies on a tree structure called convoy tree to benefit sensor collaboration among multiple nodes surrounding the target. To efficiently track the moving target, the convoy tree is dynamically configured by adding and pruning some nodes as the target moves. In [15], a pub/sub tracking method, called STUN, is proposed. The method employs a DAB tree structure to handle a large number of tracked targets. The leaf nodes are responsible for detecting the arrival and departure of a target, and only updated information will be sent to the root through intermediate nodes. By eliminating redundant message passing, the communication cost is reduced. In [16], Deviation-Avoidance Tree and Zone-based Deviation-Avoidance Tree are further proposed to reduce the communication cost of location update for in-network target tracking. The drawback of the tree-based tracking protocols is that the tree structure in WSNs is easily broken and the maintenance of the tree structure is costive.

Cluster-based tracking protocols: A static cluster-based distributed predictive tracking (DPT) protocol is presented in [7]. The protocol uses a predictive mechanism to inform the cluster head about the approaching target, and then the corresponding cluster head wakes up the most appropriate nodes immediately before the arrival of the target. DPT takes the advantages of the cluster-based structure which is especially suitable for large-scale networks. However, this scheme suffers the problem of local sensor collaboration as the target moves across or along the boundary among clusters. Wang et al. also propose a cluster-based hierarchical prediction algorithm (HPS) for energy efficient target tracking in large-scale sensor networks [13]. In contrast, a decentralized dynamic clustering protocol, relying on a static backbone of sparsely placed high-capacity sensors

(CHs), for acoustic target tracking is proposed in [5]. A high capacity sensor becomes active when its detected acoustic signal strength exceeds a predetermined threshold. The active CH then wakes up sensors in its vicinity to form a dynamic cluster for local collaboration of sensing information. An adaptive dynamic cluster-based tracking (ADCT) protocol is proposed in [6] for target tracking without the employment of high-capability sensors. The protocol dynamically selects cluster heads and wakes up nodes to construct clusters with the help of prediction algorithm as the target moves in the network. As dynamic clustering algorithms always involve the most appropriate nodes in a cluster at one time for local sensor collaboration while keeping other nodes in the sleep state, they are energy efficient for local sensor collaboration. However, the dynamic clustering process is a costive operation which repeats as the target moves. Also, it does not consider the efficient data routing issue for target tracking.
Prediction-based tracking protocols: In [17], Xu et al. propose a dual prediction-based reporting mechanism (DPR), which reduces the energy consumption by avoiding unnecessary long distance transmission between sensor nodes and the base station. A prediction-based energy saving scheme (PES) is presented in [18]. Based on predictive location, three heuristic wake-up mechanisms are introduced to balance energy consumption and tracking accuracy. An energy efficient tracking algorithm, called predict and mesh (PaM) is proposed in [19]. PaM is composed of n-step prediction, collaborative prediction and a prediction failure recovery process called mesh. Simulation shows PaM is robust against diverse motion changes. In [20], the particle filter is used to predict the target's location so that the optimal sensor nodes can be waked up to detect the target. Prediction-based algorithms suffer problems of tracking uncertainty and even the loss of the target due to unavoidable prediction errors.

3 System Model and Problem Description

In this section, we first present the system model including a network model and a sensing and communication model, then we describe the boundary problem for target tracking in cluster-based WSNs.

3.1 System Model

We assume that a large-scale WSN is formed by n static sensor nodes randomly deployed in a two dimensional area of interest. The sink node is deployed at a corner of the network. The network is organized as m clusters by using any suitable clustering algorithm in [22]. Each cluster i has n_i nodes including one cluster head and many members. Each member can communicate with its cluster head directly. Each node is aware of its own location and the information of its neighbors, but does not have global topology information.

We assume that each node is equipped with an acoustic sensor, using an identical sensing range r_s for detecting targets. A general sensing model is adopted for an acoustic sensor and the *monitoring region* of a sensor node v_i, denoted by $R(v_i, r_s)$, is the disk with the center v_i and radius r_s. A target will be detected by the sensor v_i when it appears in the monitoring region $R(v_i, r_s)$.

We assume that each node has an identical communication range r_c, which guarantees the link of any pair of nodes be symmetrical. Each node operates in the active state for transmitting packets, receiving packets and sensing the target, and in the sleep state for energy saving. Cluster heads are responsible for selecting some members to monitor the target. As the energy consumption of a node in the sleep state can be negligible, we mainly consider the energy consumption of the node in the active state as the target moves in the network.

3.2 Boundary Problem

When tracking a target in a surveillance area, multiple nodes surrounding the target collaborate to make the collected information more complete, reliable and accurate. There is no problem when the target is inside a cluster. However, when the target moves across or along the boundaries among multiple clusters, the boundary problem of local sensor collaboration occurs. The sensor collaboration is not complete and reliable because sensor nodes in the monitoring region belong to different clusters, which increases the uncertainty of the localization or even results in the loss of the target. The target is lost due to the insufficient sensing reports or the incorrect prediction of the target location collected by the network.

Fig.2 shows two cases of the boundary problem for target tracking in cluster-based WSNs. Cluster A is active whereas clusters B and C are in the sleep state. For the case in Fig.2(a), when the target moves close to the boundary of clusters, nodes a, b, c, d, e and f, which are in the monitoring region, belong to three different clusters A, B and C. Only nodes a and b can sense the target as they are active while other nodes cannot as they are in the sleep state. The network cannot successfully localize the target due to insufficient information, which may affect the prediction accuracy of the target's next position or even result in the loss of the target. For the case shown in Fig.2(b), the next location of the target is predicted in cluster B whereas the target actually moves into cluster C. Nodes in cluster B are activated in advance according to the predicted location of the target. However, none of them can sense the target since the target is

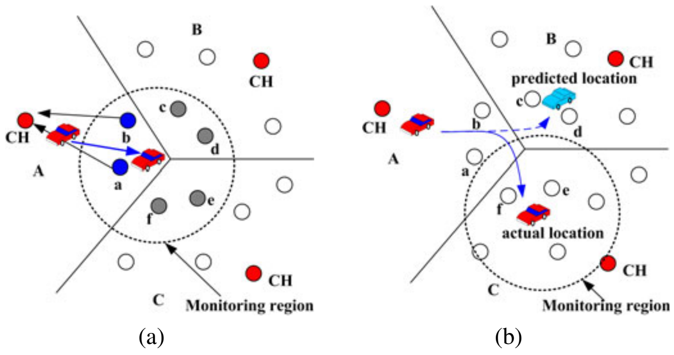


Fig. 2. The boundary problem in a cluster-based WSN: (a) high localization uncertainty due to insufficient active nodes, (b) loss of target due to incorrect prediction of the target location

moving into cluster C. Therefore, prediction error may also result in the loss of the target.

The intuitive way for solving the boundary problem is to coordinate involved clusters. When the target moves across or along the boundary of clusters, all involved clusters cooperate for local collaboration [7] [13]. For the example shown in Fig.2(a), when the target moves to the boundary among clusters, clusters A, B and C are all in the active state. Nodes that sense the target report to their cluster heads independently. Cluster heads exchange collected information with each other to make the final estimate of the target's location. However, this method is not effective as it requires both intra- and inter-cluster communications, which incurs high communication cost and large time delay.

In this paper, we propose the hybrid cluster-based target tracking protocol (HCTT) for effective target tracking in a cluster-based WSN. HCTT integrates on-demand dynamic clustering into scalable cluster-based structure with the help of *boundary nodes*. That is, when the target moves inside a cluster, the static cluster is responsible for sensor collaboration and target tracking, whereas when the target approaches the boundary among clusters, a dynamic clustering process will be triggered to avoid the boundary problem. The on-demand dynamic cluster will disappear soon after the target moves away from boundaries. Static clusters and on-demand temporary dynamic clusters alternately handle the tracking task as the target moves in the network.

4 Hybrid Cluster-Based Target Tracking Protocol (HCTT)

In this section, we first give an overview of HCTT, and then elaborate on the design and implementation of HCTT.

Fig. 3 outlines the overview of the system and illustrates the flowchart of HCTT. After being deployed, sensor nodes are organized into static clusters according to any suitable clustering algorithm. Boundary nodes in each cluster are also formed. When a target is in the network, a static cluster sensing the target wakes up to sense and track the target. When the target approaches the boundary, the boundary nodes can detect the target and an on-demand dynamic cluster will be constructed in advance before the arrival of the target for smoothly tracking the target. As the target moves, static clusters and on-demand dynamic clusters alternately manage the tracking task. Inter-cluster handoffs take place between any two consecutive clusters (i.e., the S2DIC handoff from

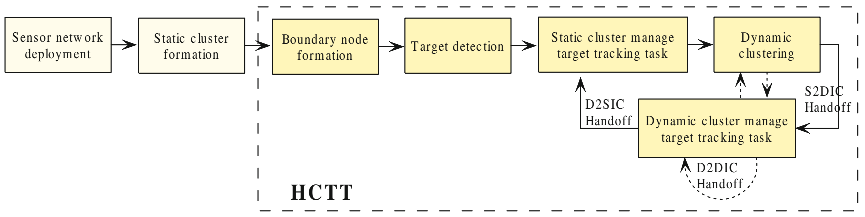


Fig. 3. Overview of hybrid cluster-based target tracking protocol

a static cluster to a dynamic cluster, the D2SIC handoff from a dynamic cluster to a static cluster, and the D2DIC handoff from an old dynamic cluster to a new dynamic cluster). Moreover, the dynamic cluster will dismiss after the target moves away from the boundary and enters another cluster. With the help of the boundary nodes, HCTT integrates the on-demand dynamic clustering into a scalable cluster-based WSN for target tracking, which facilitates the sensors' collaboration among clusters and solves the boundary problem. In the following, we elaborate on the design and implementation of three major components of HCTT, including the boundary node formation, dynamic clustering and inter-cluster handoff.

4.1 Boundary Node Formation

Dynamic clusters will be constructed when the target approaches the boundary among clusters. However, a challenging issue is how to know whether the target is approaching the boundary, especially in a fully distributed way. As sensor nodes are randomly deployed, static clusters are usually formed irregularly. It is difficult or even impossible to calculate the geometrical boundary among irregular clusters. In this paper, we use *boundary nodes* to solve this issue in a fully distributed way. A node i is a *boundary node* if there exists at least one neighbor node j such that $||l_i - l_j|| \leq r_s$ and $C(v_i) \neq C(v_j)$. Here, $C(v)$ denotes the cluster which node v belongs to. In contrast, a node is an *internal node* if it is not a boundary node. As each node is aware of its own location and its neighbor information, it checks its neighbor list to determine whether there exists a node belonging to another cluster within its sensing range. If yes, it is a boundary node; otherwise, it is an internal node. The *boundary node set* of a cluster C_i, denoted by $B(C_i)$, is formed by all the boundary nodes in C_i. The *internal node set* of a cluster C_i, denoted by $I(C_i)$, is formed by all the internal nodes in C_i.

After boundary nodes are identified, each cluster can be partitioned into three parts: safety region, boundary region, and alert region.

The *safety region* of a cluster C_i, denoted by $R_S(C_i)$, is the region in the cluster C_i that can be monitored by at least one internal node of C_i, but not by any boundary node of C_i. That is, $R_S(C_i)$ can be formulated as:

$$R_S(C_i) = \bigcup_{\forall v_i \in I(C_i)} R(v_i, r_s) - \bigcup_{\forall v_i \in B(C_i)} R(v_i, r_s) \tag{1}$$

The *boundary region* of a cluster C_i, denoted by $R_B(C_i)$, is the region that can be monitored by the boundary nodes of itself and any of its adjacent clusters at the same time. That is, $R_B(C_i)$ can be formulated as:

$$R_B(C_i) = \bigcup_{\forall v_i \in B(C_i), \forall v_j \in B(C_j), \forall C_j \neq C_i} (R(v_i, r_s) \cap R(v_j, r_s)) \tag{2}$$

The *alert region* of a cluster C_i, denoted by $R_A(C_i)$, is defined as the region that can be monitored by any boundary node of C_i, but not belongs to the boundary region of C_i. That is, $R_A(C_i)$ can be formulated as:

$$R_A(C_i) = \bigcup_{\forall v_i \in B(C_i)} R(v_i, r_s) - R_B(C_i) \tag{3}$$

Fig.4 illustrates the three different regions. The white region denotes the safety region, the light gray region denotes the alert region and the dark gray region with biases denotes the boundary region. Specifically, we have the following observations:

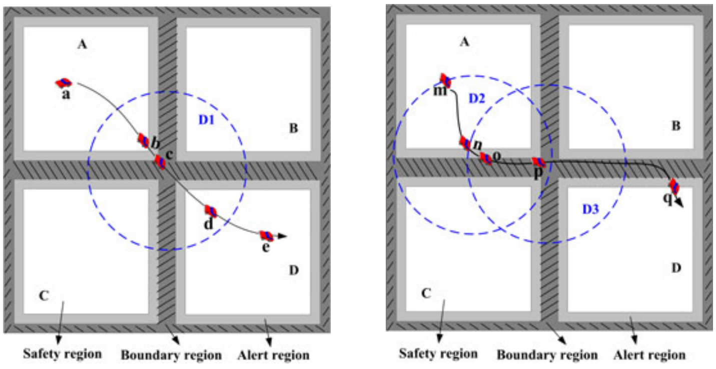
1. If the target moves into the safety region, all nodes sensing the target are internal nodes of a cluster, and vice versa.
2. If the target moves into the boundary region, there are at least two nodes belonging to different clusters that can detect it, and vice versa.
3. If the target moves into the alert region, there will be at least one boundary node can detect it and all nodes detecting the target belong to the same cluster, and vice versa.

In the following sections, we will describe in detail dynamic clustering and inter-cluster handoff based on boundary nodes and these observations.

4.2 Dynamic Clustering

Dynamic clustering is triggered on demand in order to solve the boundary problem in cluster-based WSNs. There are two cases that a dynamic cluster is needed. The first case is shown in Fig.4(a) where the current active cluster is cluster *A*. When the target moves from point *a* to point *c* via point *b*, a dynamic cluster *D*1 will be needed for smoothly tracking the target. The second case is shown in the Fig.4(b) where the current active cluster is a dynamic cluster *D*2. When the target moves along the boundaries from point *m* to point *q* via point *p*, using only one dynamic cluster cannot satisfy the target tracking requirements. Therefore, another dynamic cluster *D*3 will be needed when the target moves to point *p*.

Although the dynamic cluster must be constructed in advance before the target moves across the boundary of clusters, it is costive if the dynamic cluster is constructed too early, which not only wastes nodes' energy but also may be out of usage quickly. We can see that, if no sensing report is sent from a boundary node, the target is still in the



(a) Case 1 (b) Case 2

Fig. 4. Dynamic clustering and handoffs between clusters

safety region of the cluster; if the target is sensed by any boundary node, it is in the alert region, which means the target is closely approaching the boundary. Therefore, the on-demand dynamic clustering process will be triggered once there is at least one boundary node sensing the target.

The way of forming boundary node of dynamic cluster is different from that of static cluster. A node v_i is a *boundary node of a dynamic cluster D* if there exists at least one neighbor node $v_j \notin D$ such that $||l_i - l_j|| \leq r_s$.

The dynamic clustering process consists of three phases: leader selection, dynamic cluster construction and boundary node formation. In the leader selection phase, the active boundary node closest to the target is selected as the leader for constructing a dynamic cluster. In the dynamic cluster construction phase, the selected leader broadcasts a *recruit* message asking its neighbor nodes to join into the cluster. Each node receiving the *recruit* message establishes a new table containing the information of the dynamic cluster, e.g., the ID of the leader and the working state of the dynamic cluster. Then each node replies a *confirm* message to the leader to form the dynamic cluster. The boundary node formation phase builds the boundary nodes of the dynamic cluster by checking each node's neighbor list whether there exists a node within the sensing range that does not belong to the dynamic cluster. If a node can find at least one such kind of node, it is a boundary node of the dynamic cluster; otherwise, it is an internal node of the dynamic cluster. When the dynamic cluster is constructed, it stays in the waiting state for the coming of the target.

4.3 Inter-cluster Handoff

The tracking task should be handled by the most suitable cluster. As the target moves in the network, the task should be handed over from the pervious cluster to another most suitable one, which is taken charge of by the inter-cluster handoff process. The inter-cluster handoff occurs under the three scenarios: handoff from a static cluster to a dynamic cluster (e.g., from cluster A to cluster $D1$ as shown in Fig.4(a)), handoff from a dynamic cluster to a static cluster (e.g., from $D1$ to D as shown in Fig.4(a)), and handoff from a dynamic cluster to another dynamic cluster (e.g., from $D2$ to $D3$ as shown in Fig.4(b)). In the following part, we will describe these three types of handoff processes.

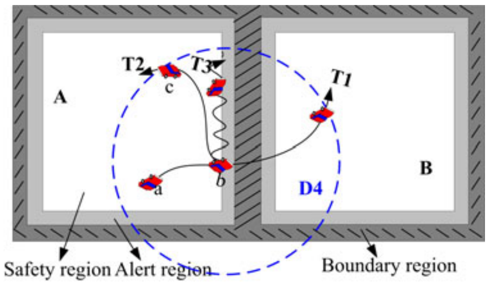


Fig. 5. Different moving trajectories of the target

Static to Dynamic Inter-Cluster Handoff (S2DIC Handoff). When the target locates within a cluster, it is tracked by the current active static cluster. As the target moves into the alert region, boundary nodes can sense the target. Upon detecting the target by a boundary node, a dynamic cluster is constructed in advance for the coming of the target. The handoff should not take place right away once the new dynamic cluster is constructed since the movement of the target is unpredictable. The target may move towards the boundary, but it may also turn around and move away from the boundary. As shown in Fig.5, when the target moves to point *b* in the alert region, a dynamic cluster $D1$ is constructed before the target moves into the boundary region. If the target's trajectory is T1, the dynamic cluster is more suitable than the active static cluster for tracking the target. However, the target may follow trajectories T2 or T3. For these trajectories, it is inappropriate to perform the handoff since the target does not move into the boundary region. Especially for the movement of T3, unnecessary dynamic clustering and handoff may take place frequently if we do not have an effective mechanism. To minimize unnecessary dynamic clustering and handoff, the S2DIC handoff starts when the target is confirmed to be in the boundary region. Before that, the dynamic cluster may not suit for target tracking.

When the dynamic cluster is constructed, some members of the current active static cluster join into the dynamic cluster. These nodes are in the waiting state while they keep sensing the target. In case any of them detects the target, it sends the sensing result to the cluster head of the dynamic cluster. When this cluster head receives the sensing result, the target is confirmed to be in the boundary region and the handoff starts. The S2DIC handoff procedure works as follows: The dynamic cluster head first sends a *request* message to the active static cluster head to ask for the handoff of the leadership. The active cluster head replies a message containing the historical estimations of the target's locations and hands the leadership over to the cluster head of the dynamic cluster. The dynamic cluster then becomes active and the new active cluster head broadcasts a *work* message to activate all its member nodes. The previous cluster head broadcasts a *sleep* message to its member nodes. Each member node not belonging to the active dynamic cluster turns into the sleep state to save energy.

If the dynamic cluster is not suitable for tracking the target, it is not needed any more and should be dismissed. As shown in Fig.5, when the target moves to point *c* following the trajectory $T2$, the dynamic cluster $D1$ is not suitable anymore and should be dismissed. The dynamic cluster dismissal procedure is as follows: If the cluster head of the dynamic cluster receives sensing results from the boundary nodes of the dynamic cluster, it confirms that the dynamic cluster is not needed any more and should be dismissed. Then the cluster head of the dynamic cluster sends a *resign* message to inform the active cluster head. After getting the acknowledged message from the active cluster head, it broadcasts a *dismiss* message to all its members. Each node receiving the *dismiss* message quits the dynamic cluster and deletes the information table built for the dynamic cluster.

Dynamic to Static Inter-Cluster Handoff (D2SIC Handoff). When a dynamic cluster is currently handling the tracking task, nodes sensing the target report their sensing results to the active dynamic cluster head. If none of sensing nodes is a boundary node of any static cluster, the target has moved away from the boundary region and has entered the safety region of a static cluster. Then the D2SIC handoff is triggered: The active cluster head sends a *handoff* message to the cluster head of the static cluster. After receiving the message, the static cluster head first replies an *acknowledge* message, then activates all nodes within its cluster. After receiving the *acknowledge* message, the previous dynamic cluster head broadcasts a *dismiss* message. Each node receiving the *dismiss* message quits the dynamic cluster and deletes the information table for the dynamic cluster.

Dynamic to Dynamic Inter-Cluster Handoff (D2DIC Handoff). When a dynamic cluster is currently active for the tracking task, the handoff condition for the S2DIC handoff may not be met as the target moves. As shown in Fig.4(b), one dynamic cluster $D2$ cannot satisfy the requirements of target tracking when the target just moves along the boundaries among clusters. In this scenario, another new dynamic cluster $D3$ will be needed for continuing the tracking task. If some boundary nodes of the active dynamic cluster sense the target, a new dynamic cluster will be constructed. Note that the D2SIC handoff has a higher priority than the D2DIC handoff as the latter one causes more cost and larger delay than the former one.

Generally speaking, the new dynamic cluster is a more preferred choice than the previous one. Therefore, the handoff takes place immediately after the new dynamic cluster is constructed: The new cluster head sends a *request* message to ask for the handoff of the leadership. The previous cluster head replies a message containing the historical estimations of the target's location to handover the leadership to the cluster head of the new dynamic cluster. After receiving the leadership, the new active cluster head broadcasts an *activate* message to inform its members to be responsible for target tracking. Each node sensing the target reports the sensing results to the active cluster head. The new cluster head then replies an *acknowledge* message to the previous dynamic cluster head. The previous dynamic cluster head broadcasts a *dismiss* message. Each node receiving the *dismiss* message quits from the dynamic cluster and deletes the information table for the dynamic cluster.

We can easily find that the process of boundary node formation has the largest computation complexity and message complexity. We assume that the node density of the network is λ. The process of boundary node formulation needs to broadcast at most n messages and compute at most $n * (\lambda \pi r_c^2)$ times. That is, for a sensor network with fixed node density and communication range, the computation complexity and message complexity are both O(n), which is proportional to the scale of the network. As for the processes of dynamic clustering and inter-cluster handoff, the computation complexity and message complexity will not be affected by the scale of the network. Therefore, HCTT is a scalable algorithm with computation complexity of O(n).

5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm through simulations. A total of 6000 sensor nodes are randomly deployed in the area of size $400 \times 400m^2$. The sensing radius of each node is fixed at $10m$. The transmission radius of each node is set to be at least twice of the sensing radius, which varies from $20m$ to $80m$. The size of a control message for constructing dynamic clusters is 10 bytes. The size of a sensing report generated by each node sensing the target is 40 bytes. The prediction error varies from 0 to r_s. We adopt the random-waypoint (RWP) model for the movement of a target: The target waits for a predetermined pause time and then chooses a destination randomly. The node moves toward this destination at a random speed in [5, 10] m/s. On arriving at the destination, the node pauses again and repeats the process. Moreover, we adopt the energy consumption model presented in [23]. To transmit a k-bit packet with a range r_c, the consumed energy is $E = E_{elec} * k + \epsilon_{friss-amp} * k * r_c^2$. Typically, $E_{elec} = 50\ nJ/bit$ and $\epsilon_{friss-amp} = 10\ pJ/bit/m^2$.

We compare the performance of HCTT with three typical target tracking protocols: DPT [7], DCTC [14] and ADCT [6]. The simulation study mainly concentrates on three performance metrics: missing ratio, sensing coverage, and energy consumption. The *missing ratio* is the probability of missing the target as the target moves in the network. The *sensing coverage* is the ratio of the number of nodes sensing the target to the number of nodes within the monitoring region. The *energy consumption* refers to the energy consumed for transmitting control messages and sensing reports.

Note that though different localization algorithms affect the performance of target tracking significantly, they are not the major concern of this paper. In order to evaluate the localization performance of the proposed algorithm, we use the metric of the sensing coverage to reflect the accuracy of estimates of the target's location since the accuracy of localization is related to the information collected from nodes surrounding the target. In other words, larger sensing coverage usually means better estimate of the target's location. It is expected that all nodes detecting the target report their results to generate reliable and accurate estimates of the target. However, many of them may not be able to sense the target successfully as they are in the sleep state due to prediction error or other factors.

5.1 Missing Ratio

Fig. 6(a) shows the effects of the prediction error on missing ratio of four algorithms. The prediction error refers to the distance difference between the estimated location

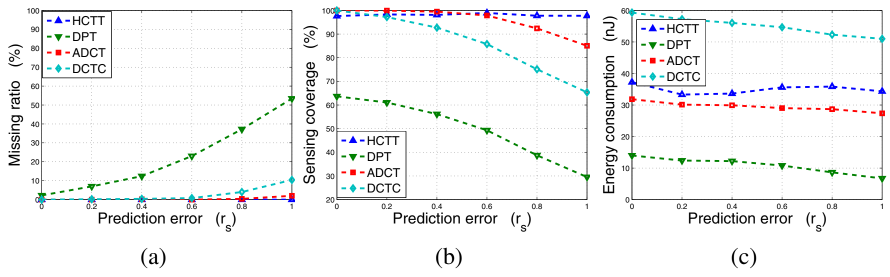


Fig. 6. Performance evaluation among DPT, DCTC, ADCT and HCTT : (a) the missing ratio vs. prediction error, (b) the sensing coverage vs. prediction error (c) the energy consumption vs. prediction error

and the real location of the target with the upper bound of the sensing range r_s. We can see that the missing ratio of DPT is much higher than that of other three algorithms. Even when the prediction error equals to zero, DPT still misses the target with high probabilities. This validates the effects of the boundary problem in cluster-based sensor networks. Besides, the missing ratio of DPT increases as the prediction error increases, which implies that the boundary problem will be deteriorated in case that the prediction mechanism does not work well for tracking the moving target. The same trend is also observed on DCTC and ADCT. In comparison, the missing ratio of HCTT keeps zero all the time. The reason is that HCTT relies on the boundary nodes instead of the prediction algorithm to trigger dynamic clusters.

5.2 Sensing Coverage

As shown in Fig.6(b), the sensing coverage of DPT, ADCT and DCTC decreases when the prediction error increases, while the sensing coverage of HCTT is not affected. That is due to the fact that increasing the prediction error results in larger deviation from the dynamic cluster to the expected cluster, which means more nodes that should be waked up to sense the target are still staying in sleep state. We can also see that the sensing coverage of DPT performs the worst among all algorithms even when the prediction error is zero. That is why DPT has the worst missing ratio among all algorithms. Besides, the sensing coverage of HCTT is nearly 100%, which means almost all nodes surrounding the target contributes to the estimate of the target's location. Therefore, the estimate of the target's location is in general more accurate and reliable than those of other three algorithms. Note that the trend of the sensing coverage in Fig.6(b) is contrary to the trend of the missing ratio in Fig.6(a).

5.3 Energy Consumption

Though HCTT outperforms DPT, DCTC, and ADCT in missing ratio and sensing coverage, it sacrifices more energy consumption. As we know, energy efficiency is also a critical factor on designing an effective target tracking protocol in sensor networks as the sensor nodes are typically powered by batteries only. To evaluate the energy efficiency of HCTT, we compare the energy consumption of HCTT to those three algorithms. The energy consumption presented here does not include the energy consumed of the failure recovery.

As shown in Fig.6(c), the energy consumption of DPT, DCTC and ADCT behaves some drops when the prediction error increases. Since increasing the prediction error results in the drops of the sensing coverage, the energy consumption drops accordingly as the number of nodes sensing the target decreases. DCTC consumes more energy than other algorithms as it relies on a tree structure which incurs more overhead than a one-hop cluster structure. In contrast, DPT consumes the least energy among all algorithms as it does not need to construct and dismiss dynamic clusters. The energy consumptions of ADCT and HCTT stand between the ones of DPT and DCTC. HCTT is not the most energy efficient algorithm as there is a tradeoff between the energy consumption and missing ratio/sensing coverage. Although DPT is the most energy efficient target tracking algorithm, it suffers the boundary problem which results in a high probability of the target loss. The performance between ADCT and HCTT are very close. However, HCTT is an algorithm designed for solving the boundary problem in cluster-based networks, which implicitly takes the advantages of the cluster-based structure. For example, data can be easily routed from a node to the sink or another node with a low delay, which is also important for target tracking. Besides, we do not consider the energy consumption of the failure recovery of these algorithms when the network loses the target. Obviously, the energy consumptions of the failure recovery of DPT, ADCT and DCTC are much more than that of HCTT, especially for that of DPT.

6 Conclusions

In this paper, we propose a novel and fully distributed mobility management scheme for effective target tracking in cluster-based WSNs. The scheme integrates on-demand dynamic clustering into scalable cluster-based WSNs with the help of boundary nodes, which facilitates the sensors' collaboration among clusters and solves the boundary problem when the target moves across or along the boundaries among clusters. The efficiency of the proposed protocol is confirmed by the simulation results.

References

1. Akyildiz, I.F., Weilian, S., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine, 102–114 (2002)
2. Lin, C.R., Gerla, M.: Adaptive clustering for mobile wireless sensor networks. IEEE Journal on Selected Areas in Communications, 1265–1275 (1997)
3. Pearlman, M.R., Haas, Z.J.: Determining the optimal configuration for the zone routing protocol. IEEE Journal on Selected Areas in Communications, 1395–1414 (1999)
4. Kozat, U.C., Kondylis, G., Ryu, B., Marina, M.K.: Virtual dynamic backbone for mobile ad hoc networks. In: Proc. of IEEE ICC, pp. 250–255 (2001)
5. Chen, W.P., Hou, J.C., Sha, L.: Dynamic clustering for acoustic target tracking in wireless sensor networks. IEEE Transactions on Mobile Computing, 258–271 (2004)
6. Yang, W.C., Fu, Z., Kim, J.H., Park, M.S.: An adaptive dynamic cluster-based protocol for target tracking in wireless sensor networks. LNCS, pp. 156–167. Springer, Heidelberg (2007)
7. Yang, H., Sikdar, B.: A protocol for tracking mobile targets using sensor networks. In: Proc. of IEEE IWSNPA, pp. 71–81 (2003)
8. Zhao, F., Shin, J., Reich, J.: Information-driven dynamic sensor collaboration for tracking applications. IEEE Signal Processing Magazine, 61–72 (2002)
9. Brooks, R.R., Griffin, C., Friedlander, D.: Self-organized distributed sensor network entity tracking. International Journal of High Performance Computer Application, 207–219 (2002)
10. He, T., Krishnamurthy, S., Luo, L., Yan, T., Gu, L., Stoleru, R., Zhou, G., Cao, Q., Vicaire, P., Stankovic, J.A., Abdelzaher, T.: Vigilnet: An integrated sensor network system for energy-efficient surveillance. ACM Transactions on Sensor Networks, 1–38 (2006)
11. He, T., Vicaire, P., Yan, T., Luo, L.Q., Gu, L., Zhou, G., Stoleru, R., Cao, Q., Stankovic, J.A., Abdelzaher, T.: Achieving real-time target tracking using wireless sensor networks. In: Proc. of IEEE RTAS, pp. 37–48 (2006)
12. Bhatti, S., Xu, J.: Survey of target tracking protocols using wireless sensor network. In: Proc. of ICWMC, pp. 110–115 (2009)

13. Wang, Z.B., Li, H.B., Shen, X.F., Sun, X.C., Wang, Z.: Tracking and predicting moving targets in hierarchical sensor networks. In: Proc. of IEEE ICNSC, pp. 1169–1174 (2008)
14. Zhang, W.S., Cao, G.H.: DCTC: Dynamic convoy tree-based collaboration for target tracking in sensor networks. IEEE Transactions on Wireless Communications, 1689–1701 (2004)
15. Kung, H.T., Vlah, D.: Efficient location tracking using sensor networks. In: Proc. of IEEE WCNC, pp. 1954–1961 (2003)
16. Lin, C.Y., Peng, W.C., Tseng, Y.C.: Efficient in-network moving object tracking in wireless sensor networks. IEEE Transactions on Mobile Computing, 1044–1056 (2006)
17. Xu, Y.Q., Winter, J.L., Lee, W.C.: Dual prediction-based reporting for object tracking sensor networks. In: Proc. of MobiQuitous, pp. 154–163 (2004)
18. Xu, Y.Q., Winter, J.L., Lee, W.C.: Prediction-based strategies for energy saving in object tracking sensor networks. In: Proc. of IEEE MDM, pp. 346–357 (2004)
19. Yang, L.Z., Feng, C., Rozenblit, J.W., Qiao, H.Y.: Adaptive tracking in distributed wireless sensor networks. In: Proc. of IEEE ECBS, pp. 103–111 (2006)
20. Wang, X., Ma, J.J., Wang, S., Bi, D.W.: Cluster-based dynamic energy management for collaborative target tracking in wireless sensor netowrks. Sensors, 1193–1215 (2007)
21. Zhong, Z.G., Zhu, T., Wang, D., He, T.: Tracking with unreliable node sequences. In: Proc. of IEEE INFOCOM, pp. 1215–1223 (2009)
22. Yu, J.Y., Chong, P.H.J.: A survey of clustering schemes for mobile ad hoc networks. IEEE Communications Surveys & Tutorials, 32–48 (2005)
23. Heinzelman, W.B., Chandrakasan, A.P., Balakrishnan, H.: An application-specific protocol architecture for wireless microsensor networks. IEEE Transactions on Wireless Communications, 660–670 (2002)

Suppressing Redundancy in Wireless Sensor Network Traffic

Rey Abe[1] and Shinichi Honiden[1,2]

[1] University of Tokyo, Tokyo, Japan
[2] National Institute of Informatics, Tokyo, Japan

Abstract. Redundancy suppression is a network traffic compression technique that, by caching recurring transmission contents at receiving nodes, avoids repeatedly sending duplicate data. Existing implementations require abundant memory both to analyze recent traffic for redundancy and to maintain the cache. Wireless sensor nodes at the same time cannot provide such resources due to hardware constraints. The diversity of protocols and traffic patterns in sensor networks furthermore makes the frequencies and proportions of redundancy in traffic unpredictable. The common practice of narrowing down search parameters based on characteristics of representative packet traces when dissecting data for redundancy thus becomes inappropriate. Such difficulties made us devise a novel protocol that conducts a probabilistic traffic analysis to identify and cache only the subset of redundant transfers that yields most traffic savings. We verified this approach to perform close enough to a solution built on exhaustive analysis and unconstrained caching to be practicable.

1 Introduction

Unnecessary data transfers waste network resources and have long been subject to studies on how to avoid them. Commonly used solutions include caching the answers to frequent data requests [1], or applying bulk compression to compact data before sending [2]. Redundancy suppression is one particular method that prevents repeated transfers of identical data over network links. The basic idea is to keep certain incoming data in memory at the receiving node. If another transmission of the same data became necessary thereafter, it can be reconstructed locally from the previously cached data instead of having it forwarded again.

This idea was first realized by Santos et al. [3] in a very simple form. Their solution kept records of recent outgoing packets at the sending node by computing a single hash value over each packet's payload content and counting repeated hash occurrences. Above a certain count threshold, it replaced the payload of the outgoing packet with its much smaller hash value. The node on the receiving end equally tracked recurring payloads and stored in a cache table the incoming data exceeding the threshold. It would then be able to replace subsequently arriving hash values by their corresponding data from local memory.

This example illustrates the key design aspects of a redundancy suppression protocol. First, traffic needs to be logged for analysis so that redundant parts can be identified. Furthermore, the sending and receiving nodes both have to agree on a policy which redundant data to cache and how the reference to such duplicate data is communicated. Recent publications [4,5,6,7,8] have focused on the problem of how to identify identical data subcontents between arbitrary data sets – instead of just matching complete packet payloads – using various fingerprinting [9] and chunking [6] methods. Such related work – discussed more thoroughly in Sect. 4 – is however aimed at optimizing protocol performance for high-speed networks and does not fit the requirements of wireless sensor networks.

At the same time – to the best of our knowledge – the application of redundancy suppression in wireless sensor networks has not yet been investigated at this point in time. It is easily motivated as reducing the amount of data transfers over the network saves transmission energy, an essential resource of battery driven sensor nodes. Furthermore, redundancy suppression is for the most part complementary to existing traffic saving methods and achieves traffic reduction independently of the many sensor network protocols already in use. We will give some illustrating examples for the above points in Sect. 2.

In this paper we present a redundancy suppression protocol that we devised with the particulars of wireless sensor networks in mind. Our solution is novel in so far as it analyzes traffic without presumptions on the features of redundancy like its frequency or granularity of occurrence. It does not limit the search space of the data analysis like existing solutions do, making it applicable for unpredictable and arbitrary traffic contents. At the same time, our protocol is geared towards finding only those overlaps in data that yield most savings when suppressed. It caches the top end fraction of such redundancies as limited by memory constraints of the nodes. On a technical level, our contribution lies in a novel utilization of chunking for redundancy analysis and its combination with a probabilistic frequency counting data structure to maintain redundancy information just accurate enough to base selective caching decisions on it. We extended an existing counting technique to deal with continuous data streams by evicting the least relevant data and by making it adjust accuracy parameters at run-time. For a detailed description of our proposed solution we refer to Sect. 3.

We evaluated our protocol in typical memory scarce environments by comparing it to a protocol that had no such resource constraints. Furthermore we investigated different policies for adapting counting parameters at runtime, so that memory was utilized efficiently enough to sufficiently identify redundant data. We verified that our idea is practicable and discuss the results in Sect. 5 before concluding in Sect. 6 with a summary and brief outlook on future work.

2 Problem Motivation

Wireless sensor networks are typically comprised of resource and energy constrained nodes[1]. In contrast, redundancy suppression protocols have so far

[1] Typical sensor nodes are battery-driven and equipped with microcontrollers that run at several tens up to hundreds of MHz and command a few tens to hundreds of KB RAM, as well as few hundred KB to a few MB of FLASH memory [10].

targeted network environments without such restricting properties. Existing solutions for instance assume the availability of several hundreds of MB RAM to keep track of and to cache traffic contents [5]. These protocols were also designed to meet high-speed bandwidth requirements by keeping computational overheads at a minimum. As such they do not match the wireless sensor network domain characterized by low-bandwidth links between nodes limited in their energy capacity. When operating such nodes, increased computational cost is readily sacrificed for efficient communication, as wireless transmissions consume several orders of magnitude more energy than computation does [10].

Furthermore, current methods rely on prevalent traffic patterns like stream-based bulk transfers found in IP networks. A preliminary analysis of representative packet traces is in such cases used to limit the search space in regard to proportions and frequencies of redundancy [5]. Sensor network traffic doesn't necessarily exhibit such homogeneity and carries unpredictable dynamics in terms of its repeating contents. For instance, periodic monitoring of temperature in a geographic region may cause identical sensor readings to be sent to a nearby data aggregation node with low recurrence frequency but high regularity. Ad-hoc event reporting on the other hand might result in irregular data bursts of repeated notifications over short time periods. Such diverse traffic features forestall the use of above optimization methods and require an unbiased traffic analysis.

Sensor networks are also increasingly being used as a general networking infrastructure for arbitrary applications as opposed to being deployed for a single specific application as in early days [10]. The variety of existing application types comes with a multitude of highly optimized network-level and application-level protocols [10]. Each of them uses arbitrary packet formats, meta-data and control messages that may cause redundant transmissions. For instance addressing schemes like attribute-based addressing significantly increase header information attached to the typically small data packets. In case of geographic addressing, numerous packets might carry the same target address region defined by delimiting border positions each represented by multiple byte length coordinates. Such examples emphasize how redundancy suppression in sensor networks should be generically applied to both meta-data and payloads, and not be protocol-specific solutions like existing header compression techniques [11] are.

One may argue that the research community in the wireless sensor domain has already explored an abundance of traffic saving methods, making the use of redundancy suppression questionable. We will as part of related work in Sect. 4 survey such efforts and point out how they are insufficient in regard to – and often also complementary to – the form of traffic redundancy discussed here.

2.1 Existing Techniques and Their Inadequacy

To suppress duplicate data transmissions, it is necessary to identify which portions of transmitted data are recurring, and thus should be cached and reconstructed at the receiver side. A naive approach to identify all such portions is comparing all byte subsets of the data against each other. The computational overhead of that approach however makes it impractical for online execution.

The approach taken by existing protocols [4,5,8] is to initially only keep and compare a selected small sample subset of the data in the form of their fingerprint hashes. If two of these so called *anchor fingerprints* were identical and thus pointed out identical data fragments between two data sets, their respective positions would be used as starting points to compare the actual data in detail to extract the complete matching data subsequence. The aspect in which protocols differ is how they choose a small set of such fingerprints that ideally pinpoint the positions of a rather large portion of redundancy with high probability.

A well established method is to divide packet data into partitions, so called *chunks* [6] for comparison. To do so, *Rabin fingerprints* [9] – not to be mistaken for the above anchor fingerprints – of a small sliding data window over the comparatively large data set are calculated first. The positions of a fixed subset of these fingerprints – for example those whose values constitute a local minimum [12] – are then selected to form boundaries of similarily sized chunks. Hashes computed over such chunks are then kept as anchor fingerprints for identifying matching partitions. Because the partitions are not set blindly – for instance by dividing data into equally sized stretches – but rather aligned to features of the underlying data, their endpoints attach to the boundaries of similar data stretches regardless of such contents' positioning inside the data set. To further reduce computational costs, often only a sampled subset of anchor fingerprints is kept for comparison [7,8]. Existing protocols combine different such partitioning and sampling methods, but they all set parameters like the partition or sample size based on experiments that preliminarily determined which settings match a good amount of redundancy in typical traffic. While such methods have proven to be practical in high-speed IP networks for their greatly reduced computational overhead, they require tuning the above parameters to prevailing traffic characteristics and storing both sampled fingerprints plus the original data for analysis.

3 Proposed Solution

Wireless sensor nodes necessitate a drastic reduction of memory space used to both identify and cache recurring data fractions. Our basic idea to meet this requirement was to selectively cache only the subset of repeating information that would yield most traffic savings when suppressed. It meant we had to find a way to identify those data fragments that had the largest size to retransmission frequency ratio without keeping the whole bulk of recent packet data for analysis and to save only thereby selected fragments in cache. Furthermore the data analysis process had to address the dynamic nature of redundancy and could not be biased by preliminarily narrowing down search parameters. We will in the following subsections present in detail how we have addressed these challenges.

3.1 Redundancy Analysis

We devised a novel approach to identifying redundancy that does not require preliminary parameter tuning or keeping the actual data for comparison, and at the same time keeps computational overheads well below the quadratic overhead of naively comparing all potential data subsets. Our method first requires several partitionings of the data each at a differently set chunk size. We used for this task the well established Winnowing chunking algorithm [12] that imposes a distance w on the consecutive cutpoints between partitions and seeks to prevent chunks from being much shorter. By running the algorithm multiple times with a step-wise decremented distance parameter w ranging from the total data size n down to 1, we attain content-aligned chunks ranging from the single full partition to a complete fragmentation into single bytes[2]. This process can be interpreted as slicing data repeatedly into increasingly finer pieces at carefully selected cutpoints. After each such cut we memorize what the pieces resulting from the cut looked like by taking a fingerprint of the produced data chunks.

Just like existing protocols, we later compare these chunk fingerprints to identify matching data fragments and count their occurrence. We do however not filter out fingerprints by sampling and we forego using such matches as anchor positions for detailed comparison of the actual data. We instead discard the data itself and rely on the assumption that the multitude of fingerprints taken from chunks at different granularities will match a large enough portion of arbitrarily long repeating data stretches. To identify the data fragments most worthy of caching, we normalize the repetition count of each data chunk by the chunk size. For a discussion of the computational overhead of our method due to the increased amount of fingerprints to calculate, as well as the empirical verification of the above assumption, we defer to Sect. 5.

3.2 Memory-Efficient Maintenance of Traffic History

While fingerprints map arbitrarily sized data to a small constant size representation, they still consume considerable memory if stored for a large number of data fragments extracted from recently transmitted packets. Thus it was necessary to devise a compact way to maintain fingerprint counts for ongoing traffic. Since our primary design idea was to cache merely a top fraction of recurring data, we were only interested in identifying a reasonably accurate set of most frequently encountered chunks instead of explicitly keeping all fingerprints and their exact counts. This led us to utilizing probabilistic frequency counting methods.

Estimating item frequencies in a data stream is a well known problem with various solutions [13,14,15] that each differ in trade-off parameters, the guarantees regarding the counting error and in memory utilization. Out of the available options, we decided to use *hCount* [16], a data structure that in recent studies has been attributed with superior performance under stringent memory limits [15]. In short, hCount is a sketch-based data structure consisting of h sets of m counters each. An item is counted by hashing its contents to one of the m counter positions in each of the h sets independently, and then incrementing these counters. Its estimated count can later be retrieved by calculating the minimum value of its corresponding counters. By varying h and m – which directly translate into

[2] The underlying Rabin fingerprints used by the Winnowing algorithm were taken based on a common 4 byte window setting, and a matching polynomial of degree 31.

the allocated number of counters – memory use can be traded off against the counting accuracy. The latter constitutes in a guaranteed error margin ϵ with probability ρ on the count taken dependent on the total sum of item counts N and the number of distinct items – also called item *cardinality* – M. We refer to the original paper [16] for the exact relations between h, m, ϵ, ρ, N and M.

In the protocol scenario at hand, we faced two challenges associated with using existing frequency counting algorithms like hCount. For one, they depend on preliminarily configuring parameters targeted at a certain count capacity and quality. In a continuos data stream however, properties like N and M are unpredictable and change dynamically. And second, these parameters also define a capacity limit of the data structure, beyond which the counting accuracy cannot be guaranteed. For an endless stream of packet data, one needs to keep track and control the contention of the data structure, for instance by evicting old entries at some point. We will now detail how we have dealt with these issues.

First, we extended hCount to allow adapting its parameters at run-time to changing M and N while preserving its counting state and error guarantees. Increasing capacity can be archived easily – and is also briefly suggested in [16] – by incrementally creating and maintaining additional hCounts instances when the current data structure runs out of capacity. We decided to add new instances – which we call *stages* from here on – by creating a hCount for double the item cardinality M the last stage had been set to. Starting with the first stage S_1 to meet a capacity M_1, the next stage S_2 would be allocated for a $M_2 = 2 * M_1$, and so on. Furthermore, we round the parameter m necessary to meet the demand of M up to a power of 2. This operation will over-allocate a stage to have a capacity beyond the desired M to some extent, but is necessary for the following functionality of also allowing sizing down the data structure. We allow compacting two stages S_a and S_b with parameters $h_a \leq h_b, m_a = 2^x, m_b = 2^{x+k} (x \geq 0, k \geq 0)$ by discarding the $h_b - h_a$ additional counter sets that S_b maintains. Furthermore for each of the remaining counter sets of S_b that now has its counterpart in S_a, we shift each counter's binary index position value in the set to the right by k bits and add its value to the counter at this shifted index position in S_b. The result is that S_a now carries the counter state it would have accumulated if all items had been added only to S_a to begin with. Any additional counting accuracy that S_b had provided is lost in the compaction.

Next we devised a method to deal with the unbounded stream of data to keep counts for. It necessitated purging content once the contention level of the data structure had been reached due to memory limits. Removing counts for specific elements from hCount is possible, but was out of the question since we were discarding all original traffic data and therefore any information on its data fragments was unavailable. The key to a solution lay in the fact that we were not interested in exact element counts, but only in identifying the most recurring data contents, and thus in discerning the recent top items ordered by their counts. Considering this requirement we decided to govern the contention level of the sketch by decreasing all its internal counters carrying positive values by an equal amount until the contention level fell below what memory limits allow for. This measure over time gradually discards the least frequent – thus least relevant – elements to make room for incoming new ones. We have furthermore implemented a dynamic memory allocation mechanism [17] for the counters to keep minimal the bits necessary to encode a count.

The above functional extensions on the one hand gave us the ability to dynamically filter and keep in the data structure only the most frequent item counts in an ongoing data stream as dictated by memory limits. On the other hand it allowed us to adjust the hCount parameters to best match the current properties N and M of the maintained item counts, so that the memory space was utilized efficiently despite dynamically changing count distributions. Our protocol conducts the following steps to do so. Upon analyzing an outgoing data packet and adding the counts of the extracted data chunks to the hCount, it measures the contention level of the data structure by calculating an estimated counting error for the currently maintained counts[3]. If that error exceeds the average count difference between the currently cached top counting elements and thus is large enough to significantly disturb the ranking of these top elements, or if memory limits have been reached due to necessary new counter bit allocations, the following adaptations are carried out. If possible, a new hCount stage is added to extend capacity. All counters are decremented until the error threshold and memory limits are within allowed range again and any stages emptied in the process are discarded. If contention levels of the hCount allow, stages are compacted by merging. The details of this workflow are listed in the pseudo code of our protocol in Fig. 1. The upshot of this process is that the data structure dynamically extends and shrinks to shift its memory allocation between bits used for each counter and the number of counters maintained.

3.3 Protocol Structure and Workflow

In our proposal the sender conducts the computationally intensive redundancy analysis for all outgoing traffic oblivious to which outgoing link is used. It synchronizes its caching decisions with the receiving nodes that only maintain the cache table for each incoming link. When an item is selected for caching, the sender adds to the outgoing packet the cache table entry index, offset position within the packet and the chunk length. The receiver inserts the denoted data portion into its cache table at the given index for later reference. From that point on, the sender replaces any further occurrences of that data fragment by its cache table index and offset position within the packet, so that the receiver can insert the missing data from its cache. The cache contents are thus completely sender controlled and imposed on the receiver. The reason for this design decision is that in our experiments we discovered that the memory requirements for hCount are much higher than the memory demand for the cache table, making this synchronized workflow more preferable to having both sender and receiver analyze traffic independently. On the more trivial side of our protocol design, we not

[3] We resorted to the error estimation method used for error correction in the original hCount. This method defines the counting error as the average count for a fixed number of virtual elements that are supposed to have a zero count [16].

```
INITIALIZATION
  instantiate hCount at sender node with a single stage // M = 64, rho = 0.99, epsilon = 0.001, size limit = 10 to 100 KB
  instantiate empty cache table at both sender and receiver with a cache table content size limit // size limit = 1 to 10 KB

FUNCTION process outgoing packet
  take all Rabin fingerprints of packet // window size = 4, polynomial degree = 31
  FOR chunk size 1 to packet byte size DO
    use Winnowing algorithm to partition packet data into chunks with window width w = chunk size
    store data chunks identified by Winnowing in array
  ENDFOR
  FOR each chunk in array DO
    IF chunk is in cache table THEN
      remove chunk data from packet
      attach cache table index of chunk and chunk offset position within the packet to the packet header
    ENDIF
    count increment = chunk size in bytes
    add the count increment to the count value of chunk in hCount // to normalize the chunk frequency by its size
    IF count value of chunk in hCount > chunk size // and thus indicates repetition
    AND IF count value of chunk in hCount is higher than value of lowest entry in cache table THEN
      insert chunk and count value as an entry into cache table
      WHILE cache table size > cache table content size limit DO
        evict cache table entry with the lowest count value
      ENDWHILE
      attach to packet header the chunk's cache table index, chunk offset position within the packet and chunk size
    ENDIF
  ENDFOR
  error estimate = average count of virtual elements // 20 elements, same as in original hCount paper
  error threshold = average count difference between chunks in cache // an approximate error margin to roughly preserve ordering
  IF error estimate > error threshold OR hCount size > hCount size limit THEN
    IF hCount size limit - current hCount size >= memory size to create a new hCount stage THEN
      instantiate a new hCount stage with double the parameter M of the currently largest stage and append it to current hCount
    ENDIF
    WHILE error estimate > error threshold OR hCount size > hCount size limit DO
      decrement all positive value counters in hCount by n // n = 1
      recalculate error estimate and error threshold
    ENDWHILE
    remove any hCount stages that had all their counter values reduced to zero in the above while loop
    IF the smaller (in terms of M) of two adjacent stages has enough capacity (sum of distinct items in both stages < M) THEN
      compact the contents of the larger stage into the smaller stage and discard the larger stage // as described in Sect. 3
    ENDIF
    IF hCount has only one stage and sum of current distinct items in the stage <  half its M THEN
      compact the contents of the stage into a new stage with half the M of the current stage and replace the current stage
    ENDIF
  ENDIF
  send packet
ENDFUNCTION

FUNCTION process incoming packet
  FOR each attached cache table index, chunk offset position and chunk size DO
    insert chunk data (packet data from chunk offset position to chunk offset + chunk size) into cache table at cache table index
    WHILE cache table size > cache table content size limit DO
      evict cache table entry with the lowest count value
    ENDWHILE
  ENDFOR
  FOR each attached cache table index and chunk offset position DO
    insert chunk data from cache table at cache table index into packet at chunk offset position
  ENDFOR
  pass packet on in the protocol stack for further processing
ENDFUNCTION
```

Fig. 1. Pseudo-code to illustrate the workflow of the proposed protocol (structure is not optimized for execution; sample parameters are given as used in our experiments)

only process packet payloads, but also header fields other than those required by the link layer MAC protocol. We assumed reliable packet delivery to be implemented based on acknowledgements or similar methods to keep cache tables in sync. Figure 1 gives a detailed functional outline of the protocol workflow.

4 Related Work

4.1 Traffic Reduction in Sensor Networks

Reducing the amount and cost of information transfers over the wireless medium to save energy consumption is a prevalent research issue in wireless sensor networking. A lot of solutions take advantage of specific sensor data qualities or application level knowledge to achieve it. A well investigated method for instance is aggregating information at intermediate nodes on their multi-hop transmission path to the base station [10]. Such in-network processing techniques however do not eliminate redundant transfers between the data source and aggregating nodes. Recent prediction-based methods transfer only the delta to predicted sensor readings [18], but even if the predictions were perfect, periodic and often redundant transmissions become necessary to distinguish node failures.

Traditional bulk compression methods have been optimized in their resource consumption to be deployed in wireless sensor networks [2]. Distributed source coding [10] further utilizes the correlation between spatially close sensor readings to encode data coming from multiple sensors. These methods however do not reflect overlaps in data that are sent over a link at consecutive points in time.

A simple variation of redundancy suppression regarding meta-data of IP packets is header compression [11]. It exploits predictable patterns in header fields based on static rules and is thus not applicable for arbitrary data redundancy. Generally speaking the variety of existing wireless sensor network protocols each incorporate protocol specific optimizations to reduce traffic, for instance by designing efficient control message interactions or caching meta-data like routing table entries [10]. The upshot of the various above solutions is that they are very application, protocol or data type specific, and that they insufficiently address data redundancy on a network link level.

4.2 Fingerprinting for Duplicate Detection

Rabin fingerprints [9] have been applied in a variety of algorithms to identify similarity or concrete overlaps in arbitrary data [3,4,7,8]. Recently, such fingerprints have furthermore been utilized for chunking data into shift-invariant partitions by orienting their boundaries on the data content [6,12]. Our protocol utilizes one such chunking method in a novel way to build a representative set of differently sized partitions for finding matching data contents.

4.3 Redundancy Suppression Protocols

The first redundancy suppression protocol for network links was proposed in [3]. This simple protocol based on matching whole packet payloads was later improved by introducing novel, often chunk-based fingerprinting methods [6,12] to detect duplicate data stretches between arbitrary data sets. A lot of effort was put into analyzing redundancy in typical Internet traffic based on packet traces to configure protocols to yield optimal results. For instance key questions that have been answered by such studies included how to set the partition size of the chunking algorithm, the sampling ratio of fingerprints, how much memory should be allocated for caching and on which nodes to place the caching endpoints [4].

Redundancy in transmission data has been exploited for numerous other purposes as well, for example to find multiple download sources for common data content [7], or to combine it with load balancing and routing aspects [4,19]. The above body of related work unfortunately has so far not addressed the specific requirements of wireless sensor networks.

4.4 Frequent Item Counting

The problem of identifying frequent items in data sets is a well known problem [13,14,15]. We extended one existing memory-efficient solution – namely the hCount [16] data structure – to cope with dynamically changing frequency distributions. Using such sketch-based probabilistic data structures for measuring, analyzing and classifying network traffic by itself is not a novel idea. Existing research however focused on speedy filtering methods for high bandwidth networks [20] or off-line processing of large trace data in a memory-efficient way [5].

Governing contents in Bloom filters and sketches has so far been dealt with by explicitly evicting elements to maintain for instance the temporally most recent elements in a data stream [14]. Our solution in contrast aims at the slightly different problem of purging least frequent elements without knowing item identities. Our hCount implementation moreover integrates an existing dynamic counter allocation technique based on partitioning [17]. To reflect the data chunk's size in its counts, we also applied the known concept of weighted counting [14].

5 Evaluation

5.1 Computational Overhead

The redundancy analysis we proposed entails a higher computational overhead than existing solutions based on a preset partitioning size and fingerprint sampling. Unlike the quadratic overhead of naively comparing all data subsets however, our method results in the following number of items to compare assuming the data is split into uniformly sized chunks for each partitioning size:

$$f(n) = \sum_{w=1}^{n} \frac{n}{w} = nH_n \quad \text{w..chunk size, n..packet size} \tag{1a}$$

$$\lim_{x\to\infty} H_n - ln(n) = \gamma \quad \gamma\text{..Euler-Mascheroni constant} \tag{1b}$$

$$\implies f(n) = O(n \log n) \tag{1c}$$

Chunking algorithms however do not partition the data absolutely uniformly and deviate a certain amount from the targeted chunk length to ensure shift invariance. The Winnowing scheme [12] we have used imposes a maximum distance w between cutpoints, but can and will also produce shorter chunks. Chunks with a length smaller than the targeted size w naturally remain valid for certain smaller w depending on how much they deviated. It makes the above overhead a worst case that is undercut in many cases. In our experiments the number of chunks big enough to justify caching actually remained below n in most cases[4]. Considering the absence of high-speed networking requirements, the small packet sizes, low bandwidths and low duty cycles typical for wireless sensor networks, we found this overhead in form of additional computational load and calculation time to be of permissible dimension.

[4] A data chunk is big enough to justify its caching, if it is larger than the number of bits necessary to encode the index of the cache table in which it will be maintained.

5.2 Empirical Evaluation

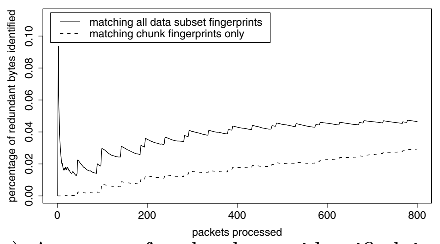
Measuring the practical gains of redundancy suppression in wireless sensor networks is a difficult task, as the redundancy in data transfers depends on application specific and potentially unpredictable traffic, but also on the presence of other traffic reducing mechanisms like the ones discussed in Sect. 4. While evaluating this aspect may be possible based on specific case studies, we did not focus on this aspect for lack of representative network trace logs and the pointlessness of constructing a scenario tailored to emphasize the benefits of our protocol.

Instead we investigated how much redundancy our protocol is able to identify compared to a full-fledged protocol without memory limitations. We simulated basic network traffic by dividing binary data files into typical packet dimensions between 64 and 128 bytes for transmission. Without any further preprocessing or encoding, we then transferred the raw data over a single link in succession. To reflect varying degrees of redundancy in network traffic, we conducted simulations with real world sensor data logs[5] carrying high degrees of correlation, but also drew on standardized *data corpora*[6] generally used for benchmarking compression algorithms. Sensor data were fed to the simulator on a per node log basis, and the data corpora evaluated for each single file of the respective data set. The input streams each spanned several hundred KB of data and resulted in several hundred to thousand packet transmissions, a traffic amount large enough to observe distinct trends in suppression performance. Retransmission overheads for cache misses caused by packet loss were not counted, as we assumed reliable transport protocols to be in place. We did however account for overheads of synchronizing and referencing cache contents as a result of our protocol design.

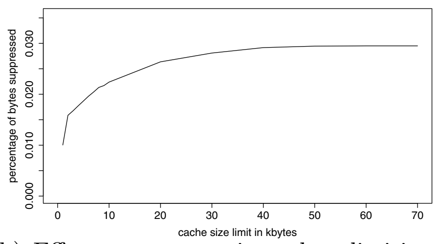
Chunking-based Redundancy Analysis. First we measured the loss of performance when matching identical data fragments by comparing chunks only. As a benchmark we calculated the computationally intensive optimal output of analyzing all potential data subsequences off-line. We could verify that the amount of redundancy identified by our method remains within a constant factor of the maximum amount substantial enough to justify its suppression. Figure 2(a) shows a sample result produced from a geophysical data set. The data in this example contained very little redundancy to begin with, a condition for which our chunking based approach delivered the worst results. In case of sensor data logs that exhibited larger amounts of redundancy – in some SensorScope data sets up to 50 percent of total traffic – we could measure results much closer to the maximum amount, sometimes as close as 90 percent of the maximum.

Efficacy of Selective Redundancy Suppression. In the next set of experiments we tested our hypothesis that redundancy has a skewed frequency distribution and thus limiting the cache contents to a fraction of the most recurring data is indeed meaningful. We evaluated how gradually limiting the capacity of

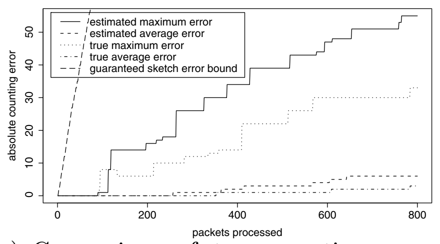
[5] The sensor logs were taken from the SensorScope [21] and ZebraNet [22] projects.
[6] The files used were taken from the Calgary Corpus and the Canterbury Corpus [23].



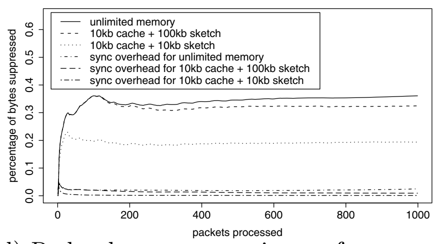
(a) Amount of redundancy identified in percentage of transferred data.



(b) Effect on suppression when limiting cache contents to a fixed memory limit.



(c) Comparison of true counting error against estimated counting error of an hCount sketch.



(d) Redundancy suppression performance under realistic memory constraints.

Fig. 2. Experimental results - (a)(b) using the geophysical data file from the Canterbury Corpus - (c)(d) using a sensor node log from the SensorScope data set

the cache table down to 1 KB affects the overall amount of traffic savings. The convex shape of the line in Fig. 2(b) representing the fraction of bytes suppressed indicates that caching a small subset of redundant data indeed yields relatively high traffic savings until the point where the line falls off steeply. Figure 2(b) again shows a sample result, but experiments with different data sets all yielded similar results, the drop-off happening between 1 KB and 10 KB in most cases.

Accuracy of Frequency Counting. Our final step consisted in limiting the memory allocated for frequency counting. The challenge lay in deciding how accurate the sketch had to be given a realistic amount of memory, a configuration that directly affects how data chunks are ranked for caching, and trades off against the amount of traffic history that can be tracked using hCount. It meant defining a count error threshold that triggered the adaptation of the sketch and how to actually measure the error. We have in Sect. 3 already foreclosed that our solution uses an estimated error measure and the average count difference between the cached elements for the threshold. This configuration – a result from experiments with different error and threshold metrics – turned out to give the best results in practice.

Since the error estimation method was only briefly mentioned in [16], we also verified that it actually approximates the real error well enough. Figure 2(c) shows a sample log of estimated and real average and maximum errors using 20 virtual elements. The nature of the above solution does not guarantee correct ranking of elements, but it turned out to balance accuracy and capacity most favorably for eventually tracking and identifying redundancy.

While we cannot present detailed results for all experiments due to space restrictions, we summarize the results as follows. We were able to archive caching performance close to a configuration with exact counting using between 10 KB and 100 KB of memory space for hCount. Figure 2(d) depicts an example data set for which 10 KB resulted in about half the performance of exact counting, and 100 KB only about 10% performance loss. The cache itself usually performed well even with an order of magnitude less memory than hCount, only a few KB to tens of KB, which in practice would further be split among the receiving neighbor nodes. Finally, we could also observe that selecting only a small top end of the recurring chunks for caching naturally kept the cache synchronization overhead within a few percent of the caching gain, making it a negligible overhead.

Conclusion of the Results. In our empirical evaluation we challenged our protocol's various design aspects. Identifying redundancy solely based on matching chunk fingerprints, selectively caching the most recurring data chunks, and basing that selection on probabilistic counts led to a practically feasible solution.

To further improve the protocol performance in the future, we see the necessity to discern at run-time whether insufficient accuracy or too short a traffic history is causing cache performance to fail, and to adapt to these possible cases more precisely and radically. For instance, if even the minimally set counting accuracy did not allow for enough history data to be kept for identifying any redundancy, giving up on chunking granularity may be a potential way to further trade off against the fraction and accuracy of repeating transfers identified.

6 Summary

In this paper we presented a first practical solution for realizing redundancy suppression in wireless sensor networks. The main challenges were to address the severe memory constraints on sensor nodes and finding a way to analyze redundancy without any assumptions on its distribution and frequency in transmitted data. We have in short devised a novel protocol by combining fingerprint based chunking, an extended sketch-based frequency counting algorithm, and by utilizing them to selectively cache an approximate top end subset of the redundant data. To attain sufficient performance we optimized memory usage of the hCount sketch by adapting its parameters and evicting its contents dynamically. In our empirical evaluation we have demonstrated the practicability of our protocol design by testing its performance under realistic memory constraints.

We are planning to further investigate open issues like the coordination of caches for multiple incoming links of a node and extending the caching scope over multiple hops. Potential functional extensions include the cross-layer coordination with multi-path routing protocols. By dividing traffic in such a way that similar data are routed along the same route we may be able to further increase the amount of traffic savings achieved by redundancy suppression.

References

1. Prabh, K.S., Abdelzaher, T.F.: Energy-conserving data cache placement in sensor networks. ACM Transactions on Sensor Networks (TOSN) 1(2), 178–203 (2005)
2. Kimura, N., Latifi, S.: A survey on data compression in wireless sensor networks. Information Technology: Coding and Computing 2, 8–13 (2005)
3. Santos, J., Wetherall, D.: Increasing effective link bandwidth by suppressing replicated data. In: Proc. of USENIX ATEC, Berkeley, USA, pp. 18–18 (1998)
4. Anand, A., Gupta, A., Akella, A., Seshan, S., Shenker, S.: Packet caches on routers: the implications of universal redundant traffic elimination. SIGCOMM Comp. Comm. Rev. 38(4), 219–230 (2008)
5. Anand, A., Muthukrishnan, C., Akella, A., Ramjee, R.: Redundancy in network traffic: findings and implications. In: Proc. of the ACM SIGMETRICS (2009)
6. Bjorner, N., Blass, A., Gurevich, Y.: Content-dependent chunking for differential compression, the local maximum approach. Journal of Comp. and Sys. Sc. (2009)
7. Pucha, H., Andersen, D.G., Kaminsky, M.: Exploiting similarity for multi-source downloads using file handprints. In: Proc. of the 4th USENIX NSDI (2007)
8. Spring, N.T., Wetherall, D.: A protocol-independent technique for eliminating redundant network traffic. SIGCOMM Comp. Comm. Rev. 30(4), 87–95 (2000)
9. Rabin, M.: Fingerprinting by random polynomials. Technical report tr-15-81, Harvard University, Department of Computer Science (1981)
10. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley & Sons, Chichester (2005)
11. Westphal, C.: Layered IP header compression for IP-enabled sensor networks. In: Proc. of the IEEE ICC, vol. 8, pp. 3542–3547 (2006)
12. Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: Proc. of the ACM SIGMOD, pp. 76–85 (2003)
13. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
14. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. Proc. of the VLDB Endowment 1(2), 1530–1541 (2008)
15. Manerikar, N., Palpanas, T.: Frequent items in streaming data: An experimental evaluation of the state-of-the-art. Data & Kn. En. 68(4) (2009)
16. Jin, C., Qian, W., Sha, C., Yu, J.X., Zhou, A.: Dynamically maintaining frequent items over a data stream. In: Proc. of the 12th ACM CIKM, pp. 287–294 (2003)
17. Aguilar-Saborit, J., Trancoso, P., Muntes-Mulero, V., Larriba-Pey, J.L.: Dynamic adaptive data structures for monitoring data streams. Data & Kn. En. 66 (2008)
18. Santini, S., Roemer, K.: An adaptive strategy for quality-based data reduction in wireless sensor networks. In: Proc. of the 3rd INSS, pp. 29–36 (2006)
19. Gupta, A., Akella, A., Seshan, S., Shenker, S., Wang, J.: Understanding and exploiting network traffic redundancy. Technical report (2007)
20. Kirsch, A., Mitzenmacher, M., Varghese, G.: Hash-based techniques for high-speed packet processing. Technical report (2008)
21. Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M., Couach, O., Parlange, M.: Sensorscope: Out-of-the-box environmental monitoring. In: Proc. of the 7th IEEE IPSN, Washington, DC, USA, pp. 332–343 (2008)
22. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L.S., Rubenstein, D.: Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In: Proc. of the 10th ASPLOS-X, New York, USA, pp. 96–107 (2002)
23. Arnold, R., Bell, T.: A corpus for the evaluation of lossless compression algorithms. In: Proc. of the 7th DCC, pp. 201–210 (1997)

Ensuring Data Storage Security against Frequency-Based Attacks in Wireless Networks

Hongbo Liu[1], Hui Wang[2], and Yingying Chen[1]

[1] Dept. of ECE, Stevens Institute of Technology
Castle Point On Hudson, Hoboken, NJ, 07030, USA
{hliu3,yingying.chen}@stevens.edu
[2] Dept. of Computer Science, Stevens Institute of Technology
Castle Point On Hudson, Hoboken, NJ, 07030, USA
hwang@cs.stevens.edu

Abstract. As wireless networks become more pervasive, the amount of the wireless data is rapidly increasing. One of the biggest challenges is how to store these data. To address this challenge, distributed data storage in wireless networks has attracted much attention recently, as it has major advantages over centralized approaches. To support the widespread adoption of distributed data storage, secure data storage must be achieved. In this work, we study the frequency-based attack, a type of attack that is different from previously well-studied ones, that exploits additional adversary knowledge to crack the encrypted data. To cope with frequency-based attacks, the straightforward 1-to-1 substitution encryption functions are not sufficient. We propose a data encryption strategy based on 1-to-n substitution via dividing and emulating techniques such that an attacker cannot derive the mapping relationship between the encrypted data and the original data based on their knowledge of domain values and their occurrence frequency. Our simulation results show that our data encryption strategy can achieve high security guarantee with low overhead.

1 Introduction

As the rapid advancement of wireless technologies has led to a future where wireless networks are becoming a part of our social life, the collected wireless data provides tremendous opportunities to support various applications ranging from environmental sensing, to infrastructure monitoring, to mobile social network analysis. However, as the amount of the wireless data is increasing, one of the biggest challenges in wireless networks is how to store these data. There are two possible ways: centralized and distributed. The traditional approach is to store the collected wireless data in a centralized manner. For example, in wireless sensor networks (WSNs) the sensing data is collected from each individual sensor and sent back to a central server for data access. However, the centralized approaches may result in performance bottlenecks of data access, and a single point of failure to both server compromise and intentional attacks.

To address these problems, distributed data storage [1,2,3,4,5] in wireless networks recently have attracted much attention. For instance, the sensed data can be stored by its type at a group of storage nodes in the network to perform data-centric storage or stored at each individual device that collects the data. The distributed data storage has major advantages over centralized approaches: storing the data on the collected wireless devices or in-network storage nodes decreases the need of constant data forwarding back to centralized places, which largely reduces the communication in the network and the energy consumption on individual devices, and consequently eliminates the existence of centralized storage and enables efficient and resilient data access. Furthermore, as wireless networks become more pervasive, new-generation wireless devices with significant memory enhancement and powerful processing capabilities are available (e.g., smart phones and laptops), making the deployment of distributed data storage not only feasible but also practical.

However, secure data storage must be achieved before widespread adoption of distributed data storage. Prior work in wireless network security has been focused on network communication security such as key management, secure localization, and intrusion detection [6, 7, 8, 9, 10]. None of these works have addressed the problem of secure distributed data storage. To fulfill the security requirements raised by the distributed data storage, recent research has started studying distributed access control, data confidentiality, and data integrity. [11] introduced a redundancy-based key distribution scheme that utilizes secret sharing to achieve a decentralized certificate authority. [12] studied to perform secure distributed data storage by developing an adaptive polynomial-based data storage scheme. [13] presented a dynamic data integrity checking scheme for verifying the consistency of data shares in a distributed manner, which is constructed based on the principle of algebraic signatures to ensure the integrity of data shares.

Most of these current research aim to provide data confidentiality, dependability, and integrity from the perspective that the adversaries will make efforts to access the data by cracking the data encryption mechanisms with little prior knowledge. None of these studies have investigated the problem of attackers cracking the data encryption by exploiting additional adversary knowledge. In particular, today with rapidly evolving adversarial activities, an attacker may possess prior knowledge about the domain values and even the exact occurrence frequencies of the data values. For instance, for the distributed data storage that stores the coordinate values of locations that people visited, the attacker may know: (1) where are the most popular locations, and (2) the fact that the frequency of these locations should be higher than that of all the other locations. Then those encrypted data values of the highest frequency must map to these popular locations. The problem gets even worse when we consider a more conservative model that the attacker knows the *exact* frequency of some or all original data values and utilizes such knowledge to crack the data encryption by matching the encrypted data values with original data values based on their frequency distribution. We call this kind of attack as *frequency-based attack*. Frequency-based attacks are especially harmful in distributed data storage. For instance, an attacker can derive the specific activities of an important officer if the attacker knows the frequency of his visited places, or a hunter can wait at specific locations of an endangered animal by possessing the knowledge of the frequency of the animal's habitation-related movements.

To cope with frequency-based attacks, apparently 1-to-1 substitution encryption function is not enough. A stronger mechanism is needed to provide more reliable data protection. However, little work has been explored to cope with frequency-based attacks. [14] has developed a secure encryption scheme to mitigate frequency-based attacks in centralized database, making it not applicable to support secure distributed data storage in wireless networks. In this paper, we propose a data encryption strategy based on 1-to-n substitution to defend against frequency-based attacks on distributed data storage in wireless networks. Our data encryption strategy aims to transform the original frequency distribution of the original data (i.e., plaintext) to a uniform distribution for the encrypted data (i.e., ciphertext) so that the attacker cannot derive the mapping relationship between the encrypted data and the original data based on their knowledge of domain values and their occurrence frequency.

In particular, we develop two techniques, *dividing* and *emulating*, to achieve the uniform distribution of encrypted data either on an individual device or across the network to cope with two types of attackers: *global frequency-based attack*, whereby the attacker only has the knowledge of the global occurrence of the data in the network, and *local frequency-based attack*, whereby the attacker's knowledge is advanced knowing about the specific occurrence frequency of the data on each individual device.

As the data is encrypted in the network for security purpose, another important issue is how to efficiently evaluate queries over these encrypted data. A naive method is to transfer all encrypted data in the network to the trusted nodes for decryption and query evaluation, which will incur tremendous communication overhead. Thus we design an efficient query evaluation procedure based on the dividing and emulating techniques for both types of point and range queries that are representative to support real-time data queries in wireless networks. Both the theoretical analysis and simulation results show that our 1-to-n substitution data encryption strategy can achieve high security guarantee, low computational cost, and efficient query processing.

The remainder of the paper is organized as follows. In Section 2, we first set up the network model, describe the attack model, and provide an overview of our strategy. We then describe our 1-to-n substitution data encryption strategy and present the details of the query evaluation procedure in Section 3. In Section 4 we discuss our simulation methodology, evaluation metrics, and results that validate our approach. Finally, we conclude in Section 5.

2 System Overview

2.1 Network Model

In our system, we consider wireless networks consisting of both static and mobile nodes, where each node represents a wireless device that can take the form of sensor, active RFID tag, laptop, or smart phone. We assume the collected data will be stored within the network at each node unless it is required to be sent to a centralized storage space for backup. By uploading data in a lazy fashion (i.e., on-demand only), distributed data storage enables real-time query evaluation and avoids frequent data transfer from the wireless devices to the centralized storage, and consequently reduces massive battery power consumption and vastly decreases the communication overhead of the network.

To prevent the misuse of the data and provide the confidentiality of the data, the data is encrypted in our network. We refer the original unencrypted data values as *plaintext* and the encrypted values as *ciphertext*. When a user queries the data in the network, all the nodes holding the data that matches the query will respond to the user by sending back the corresponding ciphertext. The user is responsible to perform the data decryption.

2.2 Attack Model

In this section, we first provide an example of frequency-based attacks. We then categorize the knowledge and behavior of the adversaries.

Example. We assume there is a set of data collected by the wireless devices where it represents the location information of animals usually appearing as shown in Table 1 (a). The original data contains the animal type and the location information.

Table 1. Data example: (a) the original data table; (b) after 1-to-1 encryption; and (c) after 1-to-n substitution encryption via dividing and emulating

Animal Type	Location
Panda	river A
Panda	river A
deer	wood B
Panda	wood C
deer	wood B
Panda	river A

(a)

Animal Type	Location
123	river A
123	river A
128	wood B
123	wood C
128	wood B
123	river A

(b)

Animal Type	Location
123	river A
123	river A
128	wood B
125	wood C
128	wood B
125	river A

(c)

As the panda is an endangered species, the information about panda's activities is sensitive and should be protected to avoid the access by poachers. The straightforward way is to use 1-to-1 encryption function to encrypt the animal type in the data set (as in Table 1 (b)). However, if a poacher has the knowledge of the animals' occurrence frequency in the dataset, there are 4 panda's record entries and 2 deer's record entries as depicted in Table 1 (a), the poacher can map the occurrence frequency of the encrypted data (e.g., 4 times of 123) to derive the corresponding animal type (e.g., panda) without decrypting the data, and consequently access the location information. In this case, the poacher will gain the sensitive information that the panda often appears at river A.

Based on the knowledge level of the original data's occurrence frequency that an adversary has, in this work we categorize the frequency-based attacks into two types: *global* and *local*.

Global Frequency-based Attack. An adversary only has the knowledge of the overall distribution of the data in the network. In particular, the adversary knows the occurrence frequency of a plaintext PT_j as $f_j = \sum_{i=1}^{N} f_{j,i}$, where $i = 1, 2, \cdots, N$ and N is the number of nodes in the network. Thus, the occurrence frequency of all the plaintext $freq(PT)$ in the network can be expressed as: $freq(PT) = \sum_{j=1}^{k} f_j$, with $j = 1, 2, \cdots, k$ and k is the number of distinctive plaintext values. However, the adversary does not have the knowledge of the detailed occurrence frequency of the data on each individual wireless device.
Local Frequency-based Attack. An adversary has the advanced knowledge of the distribution of plaintext values on each individual wireless device. Particularly, the adversary knows the occurrence frequency of a plaintext PT_j as $f_{j,i}$, with $j = 1, 2, \cdots, k$ and $i = 1, 2, \cdots, N$. k is the number of distinctive plaintext values and N is the number of nodes in the network. The local frequency-based attacks are more harmful as the attacker can derive the mapping between the encrypted data and the original data on each individual device independently.

2.3 Approach Overview

Encrypting Data via Dividing and Emulating. Based on the simple example in Table 1, we showed that simply encrypting the original sensitive data values that we want to protect by using 1-to-1 encryption functions and storing the ciphertext will result in encrypted values following the same distribution as the original plaintext values, making it easy to launch frequency-based attacks and disclosing the sensitive data to adversaries. To cope with frequency-based attacks, we propose to *divide* each plaintext value into one or more ciphertext values in such a way that regardless of the original data distribution, the target distribution remains close to flat, i.e., uniform. Furthermore, we propose *emulating* on the divided data to fit the target distribution to a uniform distribution, so that the attacker cannot uniquely crack the identity of ciphertext values, i.e., deriving the corresponding plaintext values, based on his knowledge of data frequency. Table 1 (c) shows that after applying dividing and emulating techniques, the distribution of the ciphertext values is uniform, i.e., 2 times for 123, 125, and 128 each, highly decreasing the probability for an adversary to derive the plaintext values by launching a frequency-based attack.
Coping with Attacks. Under global frequency-based attacks, our dividing and emulating techniques will exploit the global frequency distribution of plaintext values in the network to achieve uniform frequency distribution of the ciphertext values in the whole network, i.e., by examining the distribution of the encrypted values, the frequency of each data value in the network will be nearly uniform for the attacker. Whereas under local frequency-based attacks, the uniform distribution of the target ciphertext will be achieved on individual wireless device independently. Thus, the occurrence frequency of ciphertext on different nodes may be different.
Answering Data Query. We consider two types of queries on the data: *point queries* that return all data values in the network that equal to a given value, and *range queries* that return all data values in the network that fit in a range.

The query processing consists of three phases: (1) query translation at user side that transforms the original queries containing plaintext values to the ones with corresponding ciphertext values, (2) query evaluation at nodes in network that issues the translated queries on the encrypted data. For both point and range queries, since the ciphertext values are encrypted by order preserving encryption function, the ciphertext values whose plaintext values matching the original query will be returned, and (3) query post-processing at user side that decrypts the returned ciphertext values to plaintext values as the answer of the original queries.

3 Dividing and Emulating: 1-to-n Substitution Encryption

In this section, we first describe our *dividing* and *emulating* techniques that are used in 1-to-n Substitution Encryption. We then present our efficient query processing over encrypted data by using dividing and emulating techniques.

3.1 Dividing

The basic idea of dividing is that, any plaintext value of frequency f is divided into multiple ciphertext values such that the total frequency of these ciphertext values equals to f. Intuitively, if k unique plaintext values are split into $m > k$ unique ciphertext values that are of the same frequency, none of these ciphertext values can be explicitly mapped to their corresponding plaintext values by the frequency-based attack. Indeed, the *decipher probability* P that these m ciphertext values can be correctly mapped to k plaintext values by the frequency-based attack equals

$$P = \frac{1}{\binom{m-1}{k-1}}. \tag{1}$$

Number of Divided Ciphertext Values. To achieve a threshold σ of the decipher probability, for k unique plaintext values that are encrypted into m unique ciphertext values of the same frequency, they must satisfy $P = \frac{1}{\binom{m-1}{k-1}} \leq \sigma$. Intuitively, the smaller σ is, the more robust the dividing scheme is when against the frequency-based attack. Our goal is to calculate the appropriate value of m (i.e., the number of unique ciphertext values), with given σ and k. However, directly deriving m from the constraint $\frac{1}{\binom{m-1}{k-1}} \leq \sigma$ is computationally hard. Thus we consider Stirling's approximation, i.e., $m! \approx m^m e^{-m}\sqrt{2\pi m}$. We thus have:

$$P = \frac{1}{\binom{m-1}{k-1}} = \frac{1}{\frac{(m-1)!}{(k-1)!(m-k)!}} \approx \frac{1}{\frac{(m-1)^{(m-1)}\sqrt{2\pi(m-1)}}{2\pi(k-1)^{(k-1)}(m-k)^{(m-k)}\sqrt{(k-1)(m-k)}}}$$

$$\leq \frac{1}{\frac{(m-1)^{(m-1)}\sqrt{m-1}}{\sqrt{2\pi}((k-1)^{(k-1)}(m-1)^{(m-k)}\sqrt{(k-1)(m-1)})}} = (\frac{k-1}{m-1})^{(k-1)}\sqrt{2\pi(k-1)}. \tag{2}$$

As it is required that $P = \frac{1}{\binom{m-1}{k-1}} \leq \sigma$, from Equation 2, we can infer that $m \geq (k-1)(\frac{\sqrt{2\pi(k-1)}}{\sigma})^{\frac{1}{k-1}} + 1$. It is straightforward that larger m value implies the more robustness of the dividing scheme against the frequency-based attack. However, as we will discuss soon, larger m values will also result in more keys needed for encryption and decryption. To balance the trade-off between the robustness of the scheme and the cost for key management, we use

$$m = (k-1)(\frac{\sqrt{2\pi(k-1)}}{\sigma})^{\frac{1}{k-1}} + 1 \tag{3}$$

as the number of divided ciphertext values needed to achieve the required robustness of the scheme. Based on the valued m, next, we discuss how to split k unique plaintext values into m unique ciphertext values.

Dividing Factor. We define the *dividing factor* in our dividing scheme as following.

Definition 1. *Given k unique plaintext values $PT_j (1 \leq j \leq k)$ that will be encrypted as m unique ciphertext values, let $f = \Sigma_{j=1}^{k} f_j$, where f_j is the frequency of the plaintext value PT_j. Then each PT_j is encrypted as $\lceil \frac{f_j}{d} \rceil$ unique ciphertext values, where*

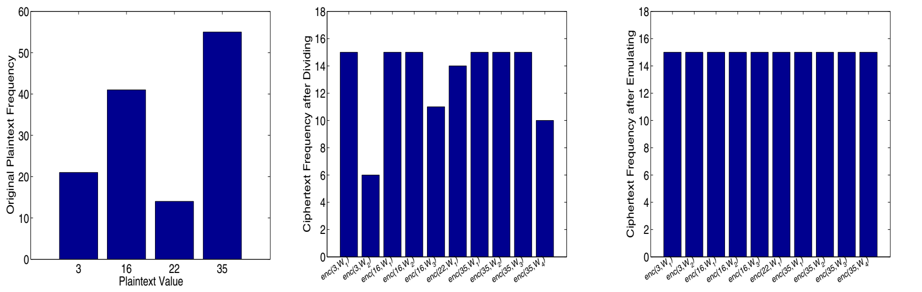
$$d = \lceil \frac{f}{m} \rceil. \tag{4}$$

We call d the dividing factor*.*

After dividing, k unique plaintext values are split into m unique ciphertext values, such that $m - \lceil \frac{md-f}{d} \rceil$ of them are of the same frequency d. If $md = f$, then all m ciphertext values are of the same frequency. Otherwise, out of these m values, there will be $\lceil \frac{md-f}{d} \rceil$ of them with frequency of $f_j - \lfloor \frac{f_j}{d} \rfloor \times d$, where f_j is the frequency of their corresponding plaintext values.

Dividing Procedure. Next, we describe the details of our dividing procedure that can achieve the goal mentioned above.
Step I. Sorting: We sort the plaintext values by their frequencies in ascending order. Let $\delta = min(PT_{j+1} - PT_j)(1 \leq j \leq k-1)$ be the minimal interval between any two successive frequency values.
Step II. Dividing: For each plaintext value PT_j, we choose t distinct random numbers $w_1, \ldots, w_t (1 \leq t \leq \lceil \frac{f_j}{d} \rceil)$ as weight values, where f_j is the frequency of PT_j, and d is the dividing factor. We require that w_2 should be unique among all the w_is, as it is needed for value decryption (More details are in Section 3.3). Then we partition f_j number of PT_j values into $\lceil \frac{f_j}{d} \rceil$ partitions, each partition containing d number of PT_j values, except the last one that contains $f_j - \lfloor \frac{f_j}{d} \rfloor \times d$ number of PT_j values. Then the PT_j value in the i-th partition $(1 \leq i \leq \lceil \frac{f_j}{d} \rceil)$ is encrypted to

$$CT_i = enc(PT_j + \sum w_i \delta), 1 \leq i \leq \lceil \frac{f_j}{d} \rceil, \tag{5}$$



(a)Original plaintext freq. (b)Ciphertext freq. after dividing (c)Ciphertext freq. after emulating

Fig. 1. Dividing and Emulating

where w_i is a distinct random number $\in (0, 1/(\lceil \frac{f_j}{d} \rceil + 1))$ (i.e., $\Sigma w_i < 1$), and *enc*() is an order-preserving encryption function [15]. More specifically, the first partition of occurrence of PT_j will be transformed to $enc(PT_j + w_1\delta)$; the l-th partition of occurrences to $enc(PT_j + \sum_{1 \leq i \leq l}(w_i\delta))$. That is to say, the l-th partition is displaced from PT_j by a fraction of the gap δ given by the sum $w_1 + w_2 + \cdots + w_l$. After dividing, there are $\lceil \frac{f_j}{d} \rceil$ number of ciphertext values $CT_1, \ldots, CT_t (1 \leq t \leq \lceil \frac{f_j}{d} \rceil)$, with their total frequencies equal to f_j.

To illustrate the results of ciphertext values by applying our dividing technique, we show a simple example as following: Given two plaintext values PT_1 and PT_2 of frequency 12 and 21, $f = 12 + 21 = 33$. Assume Equation 3 has returned $m = 5$. Then using Definition 1, the dividing factor d is calculated as 7. Based on the dividing procedure, PT_1 will be encrypted as 2 unique ciphertext values, one of frequency 7, and one of frequency 5, by using 2 unique keys; PT_2 will be encrypted as 3 unique ciphertext values, each of frequency 7.

Due to the use of order-preserving encryption function *enc*(), a nice property of the dividing scheme is that the ciphertext corresponding to different plaintext values will not straddle each other. More precisely, for any two values $PT_i < PT_j$, and for any ciphertext values CT_i^m,CT_j^n (i.e., the m-th and n-th ciphertext values of PT_i and PT_j respectively), it is necessary that $CT_i^m \leq CT_j^n$. This will enable the efficient query evaluation over the ciphertext values (More details of query evaluation will be discussed in Section 3.3).

Cost of Key Management. For each plaintext value that is divided into r unique ciphertext values, we need r unique keys. To reduce the total number of keys that are needed for dividing k unique plaintext values in the network, we allow these plaintext values share keys for dividing. Therefore, the number of keys r needed for the dividing scheme equals to $r = max_{1 \leq j \leq k} \lceil \frac{f_j}{d} \rceil$, which largely reduces the total number of unique keys during encryption.

3.2 Emulating

The dividing procedure cannot guarantee that all ciphertext values are of the same frequency. Figure 1 (a) and (b) depict an example of dividing. The 4 plaintext values 3, 16, 22, and 35 of occurrence frequency 21, 41, 14 and 55

(Figure 1 (a)) are divided into 10 unique ciphertext values (Figure 1 (b)), with the dividing factor as 15. Figure 1 (b) shows some ciphertext values, including the second ciphertext value of plaintext value 3, the third ciphertext value of plaintext value 16, the first ciphertext value of plaintext value 22, and the last ciphertext value of plaintext value 35, that are of different frequency from the other ciphertext values. These ciphertext values may face the threat that their encryption can be cracked by the frequency-based attack.

Thus, we apply *emulating* on these values, so that these ciphertext values are indistinguishable from the others by their frequencies. In particular, for these ciphertext values, they are duplicated so that their frequency also equals to d, the frequency of the other ciphertext values. Figure 1 (c) shows the results of the frequencies of these ciphertext values after emulating. Therefore, By performing emulating, these ciphertext values are indistinguishable by the frequency-based attack. However, it incurs additional space overhead for the duplicates, which is called emulating noise. There exists a trade-off between the security guarantee and the space overhead i.e., higher security guarantee may lead to more space overhead. This trade-off will be studied in details in Section 4.

To cope with both global and local frequency-based attacks, we apply the dividing and emulating encryption scheme on the plaintext values. For global frequency-based attacks, we apply the scheme on the global distribution information to achieve globally uniform frequency distribution of the ciphertext values (i.e., all unique ciphertext values in the network are of the same frequency). While for local frequency-based attacks, we exploit the dividing and emulating techniques locally on each individual device, so that the ciphertext values on each device will achieve uniform frequency distribution.

3.3 Efficient Query Processing over Encrypted Data

We assume the users issue their queries that only contain plaintext values. In this paper, we consider two types of queries: *point queries* that return all data values in the network that equal to a given value, and *range queries* that return all data values in the network that fit in a range $[l, u]$. Our goal is to translate the plaintext queries to ciphertext queries that can be applied directly on the encrypted data in the network. This mechanism has two advantages: (1) sending ciphertext queries to the network will protect the queries, especially the plaintext values in the queries, from the malicious attackers, and (2) it supports efficient query evaluation, as data decryption in the network, which in general is costly, is avoided. To achieve the goal, we design the query processing procedure that consists of three phases: query translation at the user side, query evaluation at the nodes in the network, and query post-processing at the user side. Next, we discuss the details of these three phases.

Phase-1: Query translation at user side. We assume a user can access all the auxiliary information including the weight values w_i, the gap values δ, and the order-preserving encryption function $enc()$ that are used in the dividing scheme. He/she will make use of these information to translate the plaintext queries as following.

Point queries: Given a point query $Q : V = v$, the user will translate it to Q' by following the same dividing scheme for encrypting data values in the network. In particular, the plaintext value v will be encrypted to r ciphertext values $CT_1, \ldots, CT_r$. Since these r ciphertext values follow the order that $CT_1 < CT_2 \cdots < CT_r$, the query Q will be translated to $Q' : V \in [CT_1, CT_r]$, where $CT_1 = enc(v + w_1 * \delta)$, and $CT_r = enc(v + \sum_{i=1}^{r} w_i * \delta)$. Here w_i and δ are pre-valued in the dividing scheme (see Section 3.1).
Range queries: Recall that our dividing technique performs order-preserving encryption. Thus the range query $Q : V \in [l, u]$ will be translated to another range query Q'. In particular, let w_i^l and w_i^u be the i-th weight values assigned for dividing l and u, and δ_l, δ_u be the gap values used for dividing l and u values, then the query Q will be translated to $Q' : V \in [CT_1^l, CT_r^u]$, where $CT_1^l = enc(l + w_1^l * \delta_l)$, and $CT_r^u = enc(u + \sum_{i=1}^{r} w_i^u * \delta_u)$. In other words, the plaintext range $[l, u]$ is translated to another range whose lower bound equals to the smallest divided ciphertext value of l, and upper bound equals to the largest divided ciphertext value of u.
Phase-2: Query evaluation at nodes in the network. After translation, the range query $Q' : V \in [CT_l, CT_u]$ (for both point and range plaintext queries), where CT_l and CT_u are the lower bound and upper bound ciphertext values, will be sent to the network. Each node will check whether it has any ciphertext value that satisfies the query Q', and return these ciphertext values if there is any. To ensure successful decryption in Phase-3, we require that there are at least two unique ciphertext values to be returned; if there is only one ciphertext CT value that satisfies Q', the next ciphertext value that is greater than CT will also be sent back, even though it may not satisfy Q'.
Phase-3: Query post-processing at user side. After the user receives the returned ciphertext values $CT_1, CT_2, \cdots, CT_t$ from the network, he/she will decrypt these values and obtain the plaintext values. In particular, with the knowledge of the gap values δ, he/she calculates $s_i = CT_{i+1} - CT_i$, the distance of every two successive ciphertext values (we assume $CT_1 \leq \cdots \leq CT_t$). If there exists any s_i that equals to $w_2 * \delta$, then the user deciphers CT_i as $(CT_i - w_1 * \delta)$. The reason that only w_2 is used for decryption is that if there exists any answer PT, it must satisfy that for the first and the second divided values CT_1 and CT_2 of PT, $CT_1 = PT + w_1 * \delta$, and $CT_2 = PT + (w_1 + w_2) * \delta$, thus there must exist $s_i = CT_{i+1} - CT_i$ that equals to $w_2 * \delta$. If there is no such s_i that equals to $w_2 * \delta$, then there is no answer to the queries. The success of the Phase-3 decryption is guaranteed by: (1) our design of the dividing scheme that requires that w_2 is unique among all weight values, so that is $s_i = w_2 * \delta$, and (2) our Phase-2 query evaluation procedure that requires that at least two ciphertext values (i.e., at least the first and the second divided values) should be returned.

We illustrate the query post-processing procedure through the following example: Given the plaintext values {10,11,13,14,17} with $\delta = 0.5$, and the weights $\{0.1, 0{,}3, 0.2, 0.1\}$, the divided ciphertext values will be $CT = \{10.05, 10.15, 11.05, 11.2, 11.3, 13.05, 14.05, 14.2, 14.3, 14.35, 17.05\}$. Let's consider a plaintext query $Q : V \in [13.45, 14.15]$. It is translated to the ciphertext query $Q' : V \in [13.5, 14.5]$.

Applying Q' on CT will return $\{13.05, 14.05, 14.2, 14.3, 14.35\}$. There exists two ciphertext values 14.05 and 14.2 whose distance equals to $\delta * w_2 = 0.15$. Thus the value 14.05 is deciphered as $14.05 - w_1 * \delta = 14.05 - 0.1 * 0.5 = 14$.

4 Simulation Evaluation

4.1 Metrics

To evaluate the performance of our proposed 1-to-n encryption strategy for coping with frequency-based attacks, we developed the following metrics.
Overhead by dividing. We would like to measure the additional number of ciphertext values that are introduced during the dividing process. For the case of global frequency-based attack, the overhead by dividing metric is defined as $\frac{m-k}{k}$, where m and k are the total number of distinct ciphertext and plaintext values in the network. For the case of local frequency-based attack, the overhead by dividing metric is defined as $\frac{\sum_{i=1}^{N} m_i - \sum_{i=1}^{N} k_i}{\sum_{i=1}^{N} k_i}$, where m_i and k_i are the number of distinct ciphertext and plaintext values on node $i (1 \leq i \leq N)$. Intuitively, the more the additional number of ciphertext is introduced during dividing, the more computational overhead is incurred. We will evaluate the overhead by dividing under various decipher probability for both global and local frequency-based attacks.
Overhead by emulating. In addition to evaluating the computational overhead introduced by dividing, we are interested in quantifying the additional noise amount for performing emulating in order to achieve uniform distribution of ciphertext values. We define the overhead by emulating as $\frac{S_e - S_o}{S_o}$, where S_o and S_e are the sizes of the data memory before and after emulating.

4.2 Methodology

We conducted simulation of a wireless network with multiple nodes using Matlab. Each wireless node collects the data and stores it on itself. We tested on three network sizes with number of nodes set to $N = 20, 60$ and 100 respectively. For each simulation setup, we controlled the total number of distinct plaintext values in the network to be less than or equal to 200. The occurrence frequencies of plaintext values on each wireless node are positive integer in the range of $[0, 100]$ that follows a uniform distribution. Our simulation results are the average over 100 runs for each simulation setup.

4.3 Coping with Global Frequency-Based Attacks

When coping with global frequency-based attacks, we first study the effectiveness of our scheme under various decipher probability. Figure 2 (a) and (b) present the overhead by both dividing and emulating in the network under various decipher probability when fixing the distinct plaintext values at 200. The key observation in Figure 2 (a) is that the overhead by dividing is always small (under 10%) even when the decipher probability goes to around 10^{-5}. This is encouraging as

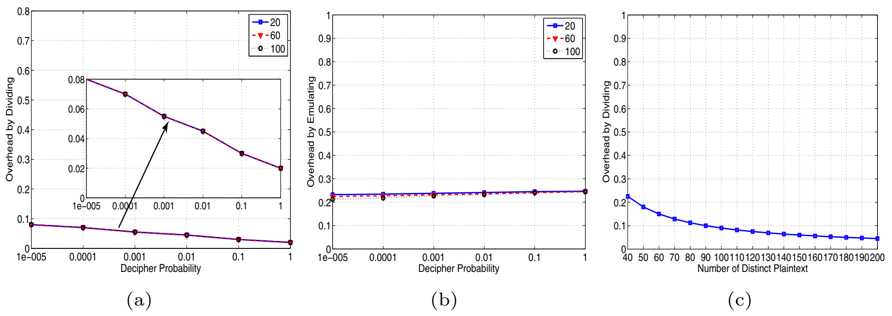


Fig. 2. Effectiveness evaluation when coping with global frequency-based attacks: (a) and (b) overhead by dividing and emulating under various decipher probability when fixing the distinct plaintext values at 200; (c) overhead by dividing under various distinct plaintext values when fixing the decipher probability to 0.01 and $N = 60$

it indicates that our scheme can achieve a robust security guarantee under global frequency-based attacks with little overhead incurred by the dividing technique. The other observation is that the curves for different network sizes overlap. This is because the number of ciphertext values required for dividing does not depend on the number of nodes in the network.

Additionally, we observed that the overhead by emulating is not sensitive to the decipher probability. It goes up slightly from 22% to 25% as the decipher probability increases, which is not significant. We further found that the overhead by emulating does not change with the network size. These discoveries suggest that our scheme is robust in terms of the overhead noise produced by the emulating process when achieving a high security guarantee under various network sizes.

We further investigated the overhead by dividing when varying the number of distinct plaintext values in the network. Figure 2 (c) depicts the overhead by dividing as a function of the number of distinct plaintext values when the decipher probability $P = 0.01$ and the network size $N = 60$. We found that the overhead by dividing is sensitive to the number of distinct plaintext values in the network. Particularly, the overhead by dividing decreases from 22% to 8% when the number of distinct plaintext values increases from 40 to 200 in the network. This indicates that the more distinct plaintext values exist in the network, the less additional computational cost is incurred when using our encryption scheme.

4.4 Coping with Local Frequency-Based Attacks

Next, we turn to examine the performance of our scheme under local frequency-based attacks. Figure 3 presents the total overhead introduced in the network by applying our decryption scheme under local frequency-based attacks when fixing the total number of distinct plaintext values at 200 (however, the number of distinct plaintext values on each node is less or equal to 200). As shown in Figure 3 (a), we observed that the overhead by dividing is comparable to that under global

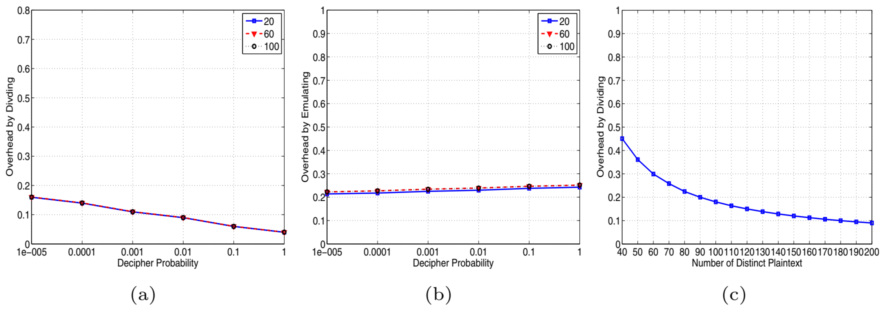


Fig. 3. Effectiveness evaluation when coping with local frequency-based attacks: (a) and (b) overhead by dividing and emulating under various decipher probability when fixing the distinct plaintext values at 200; (c) overhead by dividing under various distinct plaintext values when fixing the decipher probability to 0.01 and $N = 60$

frequency-based attacks. In particular, the overhead by dividing decreases, from 16% to 4%, as the decipher probability increases in a large range from 10^{-5} to 1. Furthermore, the overhead introduced by emulating when varying the decipher probability is presented in Figure 3 (b). We found that the overhead by emulating is not sensitive to the changes of decipher probability and having a slightly increasing trend from 22% to 25% when the decipher probability increases (from 10^{-5} to 1).

These observations are inline with those found in global frequency-based attacks, and indicating that our scheme does not require more overhead when coping with local frequency-based attacks than that under global frequency-based attacks. Additionally, we observed that the overhead do not increase as the network size increases, suggesting that both the additional computational cost and memory overhead introduced by our scheme are stable and will not vary with the network sizes.

Finally, we look at the overhead by dividing as a function of the number of distinct plaintext values in Figure 3 (c) when $P = 0.01$ and $N = 60$. The observation of the declining trend from 45% to 9% as the number of distinct plaintext values increases from 40 to 200 in the network indicates that under local frequency-based attacks, our scheme is sensitive to the number of distinct plaintext values in the network. Furthermore, the overhead by dividing is larger than the corresponding ones under global frequency-based attacks. This is based on our observation that to achieve a given decipher probability, smaller number of distinct plaintext values will require relatively more distinct ciphertext values. Since the uniform distribution of ciphertext values is achieved on each individual node when coping with local frequency-based attacks as opposed to that achieved in the network level under global frequency-based attacks. The overhead ratio by dividing of local frequency-based attacks is higher than that of global frequency-based attacks.

5 Conclusion

In this paper, we proposed a secure data storage scheme that can effectively defend against frequency-based attacks in wireless networks. We considered a sophisticated attack model that the attackers possess the knowledge of frequencies of the original data in the network and utilize such knowledge to decipher the encryption on these data. To cope with such frequency-based attacks, we designed a novel 1-to-n encryption scheme that utilizes our proposed dividing and emulating techniques. We showed that our dividing and emulating techniques not only provide robust security guarantee against frequency-based attacks but also support efficient query evaluation over encrypted data. Our extensive simulation results confirmed the effectiveness and efficiency of our approach.

References

1. Pietro, R.D., Mancini, L.V., Soriente, C., Spognardi, A., Tsudik, G.: Catch me (if you can): Data survival in unattended sensor networks. In: Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom) (2008)
2. Girao, J., Westhoff, D., Mykletun, E., Araki, T.: Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. Ad Hoc Networks 5, 1073–1089 (2007)
3. Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., Estrin, D.: Data-centric storage in sensornets. ACM SIGCOMM Computer Communication Review archive 33 (2003)
4. Ghose, A., Grossklags, J., Chuang, J.: Resilient data-centric storage in wireless adhoc sensor networks. In: Chen, M.-S., Chrysanthis, P.K., Sloman, M., Zaslavsky, A. (eds.) MDM 2003. LNCS, vol. 2574, pp. 45–62. Springer, Heidelberg (2003)
5. Shao, M., Zhu, S., Zhang, W., Cao, G.: pdcs: Security and privacy support for datacentric sensor networks. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM) (2007)
6. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.: Spins: security protocols for sensor netowrks. In: 7th ACM International Conference on Mobile Computing and Networking (2001)
7. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: 10th ACM Conference on Computer and Communications Security (2003)
8. Capkun, S., Hubaux, J.P.: Secure positioning of wireless devices with application to sensor networks. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), pp. 1917–1928 (2005)
9. Chen, Y., Trappe, W., Martin, R.P.: Detecting and localizing wirelss spoofing attacks. In: Proceedings of the Fourth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON) (May 2007)

10. Yang, J., Chen, Y., Trappe, W.: Detecting sybil attacks in wireless and sensor networks using cluster analysis. In: The Fourth IEEE International Workshop on Wireless and Sensor Networks Security (IEEE WSNS) (2008)
11. Joshi, D., Namuduri, K., Pendse, R.: Secure, redundant and fully distributed key management scheme for mobile ad hoc networks: an analysis. EURASIP Journal Wireless Communnication Networks (4), 579–589 (2005)
12. Nalin, S., Yang, C., Zhang, W.: Securing distributed data storage and retrieval in sensor networks. In: 5th Pervasive Computing and Communications (2007)
13. Wang, Q., Ren, K., Lou, W., Zhang, Y.: Dependable and secure sensor data storage with dynamic integrity assurance. In: 28th IEEE International Conference on Computer Communications (2009)
14. Wang, H., Lakshmanan, L.V.: Efficient secure query evaluation over encrypted xml database. In: 32nd International Conference on Very Large Data Bases (2006)
15. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2010)

Time-Critical Data Delivery in Wireless Sensor Networks

Petcharat Suriyachai, James Brown, and Utz Roedig

Lancaster University, UK
{p.suriyachai,j.brown,u.roedig}@lancaster.ac.uk

Abstract. A number of wireless sensor network (WSN) applications demand timely data delivery. However, existing WSNs are designed to conserve energy and not to support timely data transmission. This paper shows how WSNs can be dimensioned, deployed and operated such that both reliable and timely data delivery is ensured while scarce energy is preserved. The presented solution employs a novel Medium Access Control (MAC) protocol that incorporates topology control mechanisms to ensure timely data delivery and reliability control mechanisms to deal with inherently fluctuating wireless links. An industrial process automation and control scenario at an oil refinery in Portugal is used to define protocol requirements. The paper details a TinyOS implementation of the protocol and its evaluation in a testbed. Under high traffic load, the protocol delivers 100% of data in time using a maximum node duty cycle as little as 2.48%. In an idle network a maximum node duty cycle of only 0.62% is achieved. This proposed protocol is thus an extremely energy efficient solution for time-critical data delivery.

1 Introduction

Future application areas of wireless sensor networks (WSNs) may include industrial process automation, aircraft control systems or traffic management systems. In such systems, the WSN is part of a control loop, and predictable network performance in terms of message transfer delay and reliability is required.

To construct functioning control loops, it is necessary to ensure that sensor data is transported within a given time bound D. If too much sensor data arrives late or not at all due to losses on the wireless links, the control loop is unable to function. Data needs to be transported to the decision point (normally the sink) within a given time D_S. Thereafter, a command needs to be transferred within a given time D_A from the sink to an actuator. D_A might be zero if the actuator is located at the sink. Data arriving late cannot be used in the decision process and has to be considered lost. Early data delivery is acceptable but not desirable as it implies that resources are used inefficiently. Instead of delivering data early, the network may spend these resources to improve transmission reliability or energy consumption of nodes.

This paper presents a MAC protocol that is capable of supporting applications with the previously outlined communication requirements. The novel TDMA-based Medium Access Control (MAC) protocol incorporates routing, topology control and reliability control mechanisms to achieve the set goals. This protocol assumes that a small and relatively static network with predictable traffic patterns is deployed in a known environment. These assumptions hold true for the target area of industrial process automation and control. A network wide transmission schedule is determined before network deployment, ensuring collision free and timely data delivery. During network operation reliability control mechanisms are employed to cope with fluctuating characteristics of the wireless channel.

Designing a protocol with the outlined performance goals in wireless communication is very challenging. The protocol must be able to provide stable data transport performance using non-deterministic wireless links. Such performance is the main design goal, but energy consumption cannot be neglected as a reasonable network lifetime must be achieved. Moreover, the protocol should not be too complex so that it can be debugged easily and executed on resource constraint nodes. Finally, the protocol should allow flexibility to enable the addition of new nodes and minor topology changes.

The paper gives a description of a real deployment scenario for which the protocol is designed: industrial process automation and control for an oil refinery in Sines, Portugal. This scenario and its requirements motivate the development of the communication protocol described and evaluated in this paper. The main contributions of the paper are:

- GinMAC: A novel TDMA-based MAC protocol called GinMAC is presented, and details of its implementation on TinyOS for TelosB nodes are provided.
- GinMAC Evaluation: The testbed evaluation validates that GinMAC can support the target industrial process control and automation applications. The delay and reliability requirements are met while achieving extremely low duty cycles, leading to an acceptable network lifetime.
- Delay Conform Reliability Control: GinMAC incorporates a variety of strategies to deal with inherently non-deterministic wireless links while adhering to the given delay requirements. The effectiveness and associated energy cost of the different reliability control strategies are evaluated and discussed.

The paper is organized as follows. Section 2 describes an industrial process control and automation scenario which demands a protocol such as GinMAC. The analysis of this real application scenario defines our network assumptions and requirements. Section 3 provides a detailed description of GinMAC. Section 4 presents an evaluation of GinMAC. Section 5 reports on related work dealing with the deployment of time-critical sensor networks. Section 6 concludes the paper.

2 Assumptions, Requirements and Problem Definition

Most industrial process control and automation applications can be considered time-critical. In these scenarios, control loops are mapped onto wireless sensor networks. For such control loops to function, data needs to be delivered reliably and within a given time bound [1]. Furthermore, it might be necessary to react upon the received data, and a command needs to be delivered reliably and within a given time bound from the sink to an actuator in the sensor network. As WSNs are considered to be relatively unreliable, they so far have seen little use for such tasks. However, given the large cost savings due to a wireless deployment in industrial settings, the feasibility of wireless process control and automation is recently investigated.

Fig. 1. The GALP oil refinery in Sines, Portugal and deployed sensors/actuators

In this section, we use the GALP oil refinery at Sines, Portugal as an example scenario for industrial process control and automation. Oil refineries employ a large number of sensors to monitor their processes. Cabling cost is the main expense in installing such systems. Moreover, cabled control systems are inflexible as installation time is needed whenever production processes must be added or modified. Thus, it is desirable to utilize WSNs as base for such control systems to circumvent this hurdle.

We visited the GALP oil refinery and spoke to technicians who have installed and maintained the currently cabled process automation and control systems to identify system requirements of a WSN.

2.1 The GALP Refinery Application Scenario

The GALP oil refinery at Sines, Portugal is a complex industrial facility that includes a wide range of processing, and such processing needs careful monitoring and control of operations. There are currently 35000 sensors (some shown in Figure 1) and actuators in use in the refinery to perform real-time monitoring of industrial operations such as leakage detection and measurement of pressure in the pipes, fluid levels and of the overall environment. The monitoring of the environment in a refinery provides essential information to ensure the good health of both the refinery and its production processes. In such a typical section of the distribution system, a number of system conditions need to be closely monitored by sensors:

- Pressure is monitored within each pipe not only for safety reasons to keep pipe pressure within pipe tolerances but also to detect leakage and derive flow information. Pressure is usually measured in Pascals. A typical pressure sensor can use a $32bit$ sample size. Pressure is typical sampled at a frequency of $f = 1Hz$.
- Temperature is monitored within each pipe and tank for safety reasons. In addition, it is measured in degrees Celsius (C) at a typical frequency of $f = 1Hz$ with a sample size of $32bits$.
- Shut-off valves are monitored to ensure that they are in functioning condition when needed, and they are periodically partially closed to check that the closing mechanism is operational. Valve status can be encoded as $8bit$ value. The value is sent periodically ($f = 3 \cdot 10^{-3}Hz$), which can be adjusted by plant technicians.

The plant also has a number of actuator systems that can be directly controlled from the refineries control center. These actuators include the following:

- Shut-off valves are integrated into pipes and are used to interrupt product flow during day to day operations and in the case of emergency.
- Pumps can operate at different speeds to increase or decrease the pressure and thus flow of product through the piping system.

Sensor samples should be transported to the control center within a few seconds. The exact delay requirement depends on the production process, but delay bounds of up to $D_S = 1s$ are required. Likewise, actuators must be reached within a few seconds, and a delay bound of $D_A = 1s$ is a typical value. All data losses should be avoided, but small loss rates in the order of 0.01% are deemed to be acceptable within control loops.

2.2 Assumptions and Requirements

The example refinery deployment can be used to determine which assumptions can be made and which requirements a WSN has to fulfill. The requirements and assumptions are derived from the investigated refinery process control and automation application. However, we are confident that other large scale production facilities have very similar structures, and therefore the communication protocol presented in this paper is valid for not only the outlined application scenario but also other similar scenarios.

Deployment: Sensors and actuators are installed at carefully chosen positions. It is possible to change their location slightly by a few meters during deployment. During operation nodes are static and do not move. In addition, the physical environment is fairly static as well. Thus, sensor nodes equipped with wireless radios can be positioned such that reasonable connectivity to a neighboring node is given. It is also possible to determine worst-case and best-case link reliability during deployment phase of the network.

Maintenance: Most sensors can be mains powered, but some sensors may still be battery powered. Thus, energy consumption patterns are important but not necessary for all nodes. Sometimes additional sensors are deployed only for a few days to collect additional data. The network must consequently provide some degree of flexibility.

Topology: The refinery has a large number of in-field control stations which are interconnected by a cabled network and have a power supply. All in-field control stations forward data to the main refinery control room. The in-field control stations are currently used to aggregate data from cabled sensors and actuators in their vicinity. If a WSN is used, the in-field control station can act as a sink to which sensors and actuators in the vicinity connect via wireless communication. An in-field control station manages up to $N = 25$ nodes. Most nodes will be located one hop from the sink (determined by on-site experiment) and at most a distance of 3 hops will be observed. All nodes controlled by one in-field control station can use one frequency, while neighboring in-field control stations use another transmission frequency. It is feasible to put nodes in the refinery in small clusters that are interconnected via the in-field control stations by a cabled network.

Traffic: Nodes might report data frequently with a relatively high data rate (up to $f = 1Hz$). However, the packet payload can be considered to be quite small. Data is expected to reach the sink within a given time bound D_S. This time bound can be expected to be in the order of a few seconds; the tightest bound of $D_S = 1s$ must be considered. The commands sent from the sink to actuators must arrive within a given time bound D_A

which can be as low as $D_A = 1s$. The sink might also send commands to sensor node to set sampling frequencies and to issue other configuration commands. These command messages may not need to be delivered within a given delay bound. The traffic patterns of the nodes in the network are reasonably known at deployment time.

3 Time-Critical Data Delivery with GinMAC

With the the target application domain and its requirements in mind, we choose to implement the following main features in GinMAC:

1. *Off-line Dimensioning:* Traffic patterns and channel characteristics are known before network deployment. Thus, complex protocol operations such as calculation of the transmission schedule is performed off-line and before network deployment.
2. *Exclusive TDMA:* Only a small number of nodes ($N \leq 25$) need to be accommodated. Nodes are placed close to each other, and a high level of interference can be expected. A TDMA schedule with exclusive slot usage is consequently selected.
3. *Delay Conform Reliability Control:* The protocol must support delay bounds of D_S and D_A while achieving very high data transport reliability. Hence, all available flexibility in transport delays is used to improve reliability, given that our energy consumption target permits.

3.1 Off-Line Dimensioning

A network dimensioning process is carried out before the network is deployed. The input for the dimensioning process are network and application characteristics that are known before deployment. The output of the dimensioning process is a TDMA schedule with frame length F that each node has to follow.

The GinMAC TDMA frame consists of three types of slots: *basic slots*, *additional slots* and *unused slots*. First, the frame contains a number of *basic slots* which are selected such that within frame length F each sensor can forward one message to the sink and the sink can transmit one message to each actuator. Second, the GinMAC frame uses *additional slots* to improve transmission reliability. Finally, the frame may contain *unused slots* which are purely used to improve the duty cycle of nodes.

The above types of slots within the GinMAC frame must be designed such that the delay ($F < min\{D_S, D_A\}$), reliability and energy consumption requirements are met. However, it may not always be possible to find a frame that simultaneously fulfills all three requirements. If that is the case, some dimensioning assumptions must be relaxed.

To determine the number of *basic slots* required in a GinMAC frame, a topology envelope is assumed. This topology envelope is specified as a tree rooted at the sink and described by the parameters: maximum hop distance H and fan-out degrees O_h $(0 \leq h \leq H)$ at each tree level h; we define $O_0 = 1$. The topology envelope can accommodate a maximum number of $N^{max} = \sum_{n=1}^{H} \prod_{m=1}^{n} O_m$ nodes. However, in the actual deployment a number of nodes $N \leq N^{max}$ may be used. Nodes in the later deployment can take any place in the network and even move as long as the resulting deployed topology stays within this topology envelope. The maximum number of sensor nodes N_S^{max} and actuator nodes N_A^{max} (with $N^{max} = N_S^{max} + N_A^{max}$) must also be known.

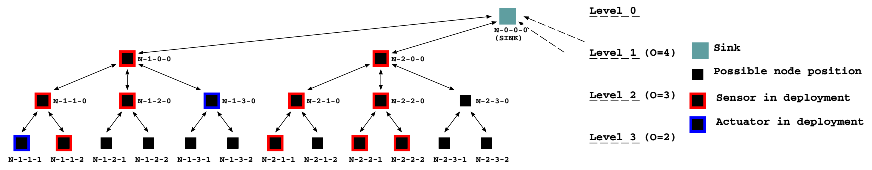


Fig. 2. Example topology with $N_A = 2$ actuators and $N_S = 10$ sensors

To determine the number of *additional slots* needed for reliability control, the worst-case link characteristics in the deployment area must be known. As the network is deployed in a known environment, it is possible to determine this value by measurement. The configuration of *basic* and *additional* slots determines an energy consumption baseline of nodes. Adding *unused slots* within the GinMAC frame can improve upon this baseline. Next, we present how to obtain the GinMAC frame configuration.

3.2 TDMA Schedule and Reliability Control

GinMAC uses TDMA slots whose size is fixed and large enough to accommodate a data transmission of a maximum length and an acknowledgement from a receiver. Moreover, these slots are used exclusively; a slot used by one node cannot be re-used by other nodes in the network. The protocol therefore does not scale to networks with many nodes. However, as described in Section 2, this scalability restriction is not an issue in the target application scenario. Furthermore, nodes are close together, resulting in high levels of interference that would limit potential slot re-usage. Finally, exclusive slot usage allows us to construct a protocol which is relatively simple to implement.

Basic Slots: The topology envelope, which is defined by the maximum hop distance H and fan-out degrees O_h, is used to calculate the basic slot schedule within the GinMAC frame. An example topology envelope for $H = 3$ and $O_1 = 4$, $O_2 = 3$, $O_3 = 2$ is shown in Figure 2; recall that we define $O_0 = 1$. Basic slots S_B are dimensioned under the assumption that all positions in the topology envelope will be occupied by nodes.

The basic slots S_B accommodate two different traffic flows. First, a number of basic slots S_B^{up} are required to accommodate traffic flowing from all nodes to the sink; we assume actuators might be used for sensing as well. Second, a number of basic slots S_B^{down} are required to accommodate traffic flowing from the sink to actuators ($S_B = S_B^{up} + S_B^{down}$). A leaf node (level H) in the tree requires one basic TDMA slot within F to forward data to its parent node. This parent node requires a slot for each child node plus one slot for its own data for forwarding to its parent. Its slots must be located after the slots used by its children to ensure that data can travel within one GinMAC frame through the tree to the sink. The allocation of the transmission slots for the previously given topology is depicted in Figure 3. The total number of slots in F needed to forward data to the sink S_B^{up} can be calculated as follows. A node at tree level h requires $S_{B,h}^{up} = O_{h+1} \cdot S_{B,h+1}^{up} + 1$ with $S_{B,H}^{up} = 1$. Consequently, S_B^{up} can be calculated as:

$$S_B^{up} = \sum_{h=1}^{H} S_{B,h}^{up} \cdot \prod_{i=1}^{h} O_i \qquad (1)$$

In the topology shown in Figure 2, $S_B^{up} = 100$ is required.

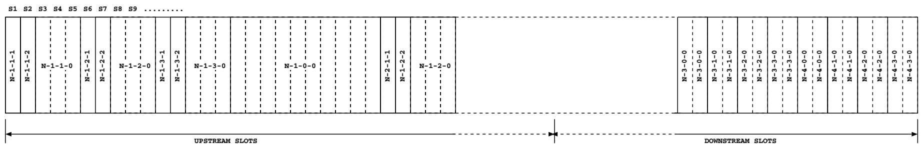


Fig. 3. Transmission slot allocation for the topology shown in Figure 2

The sink must be able to send a data packet to each actuator within one GinMAC frame. Thus, the sink requires some slots for these actuators. The slot allocation for nodes at level h is the minimum between the maximum number of actuators (N_A^{max}) in the network and the number of nodes below this level h. The required number of downstream slots $S_{B,h}^{down}$ for each node at level h can be calculated as $S_{B,h}^{down} = \min \{N_A^{max}, \sum_{i=h+1}^{H} \prod_{j=0}^{i} O_j\}$ with $S_{B,H}^{down} = 0$. Hence, S_B^{down} can be calculated as:

$$S_B^{down} = \sum_{h=0}^{H-1} S_{B,h}^{down} \cdot \prod_{i=0}^{h} O_i \tag{2}$$

In the topology shown in Figure 2 where there is a maximum of $N_A^{max} = 2$ actuators in the network, $S_B^{down} = 34$ is therefore required.

There could be *configuration commands* from the sink to nodes. In this case, we assume that such commands are not time-critical. They thus can be broadcasted when there is no *actuation command* to send, reusing the slots provisioned for the actuators.

Additional Slots: The basic slots can only assure data transport if no messages in the network are lost due to an erroneous wireless channel. However, in an industrial setting this erroneous channel is unavoidable, and consequently our protocol must provide some transmission redundancy. GinMAC employs the so-called *additional slots* S_A to implement temporal and spatial transmission diversity.

To determine the number of additional slots, we first need to choose a worst-case link reliability that GinMAC will support; the definition of reliability is elaborated in the next paragraph. The deployed system will form a topology that fits into the topology envelope and uses only links with reliability better than the selected worst-case link reliability. These links are called *good links*. Pre-deployment measurements are used to determine a reasonable value for the worst-case link reliability such that enough good links are available for topology formation. GinMAC monitors link reliability during operation and removes the links whose reliability becomes lower than the worst-case reliability threshold.

We propose two methods for specifying *good links*. The first method simply uses the Packet Reception Rate (PRR). A good link is defined as having a PRR above a specific threshold. The second method applies burst lengths to define worst-case link reliability. A good link must not have more than B_{max} consecutive transmission errors and must provide at least B_{min} consecutive successful transmissions between two bursts. A recent study has shown that this definition captures link quality better than PRR [3].

Temporal Transmission Diversity: In a scenario where good links can be characterized with short B_{max} and long B_{min}, it is possible to efficiently add additional retransmission slots on the same link to deal with losses. Consider node N-1-1-0 in the example shown in Figure 2, $B_{max} = 2$ and $B_{min} = 2$. The node requires 3 basic slots for upstream transmissions, and in a worst case any 2 of the 3 transmissions might be lost. However, if 4 additional transmission slots are allocated, all 3 packets are guaranteed to be delivered within the 7 slots provided that the channel conforms to chosen B_{max} and B_{min}. The number of necessary additional slots $S_{A,h}$ per node at level h can be calculated as:

$$S_{A,h} = \left\lceil \frac{S_{B,h}}{B_{min}} \right\rceil \cdot B_{max} \tag{3}$$

The additional upstream and downstream slots are added in the schedule directly after the respective basic slots for each direction. If a node fails to transmit data in a basic slot, it can use an additional slot for a retransmission. If links are good links as defined by B_{max} and B_{min}, all messages are delivered successfully and in time.

There might be scenarios where only links can be found that have a relatively high PRR but simultaneously have a long B_{max} and short B_{min}. In this case, these links are generally of good quality, but sometimes transmissions are impossible for extended periods of time. Thus, $S_{A,h}$ would become excessively large, and the delay target may be violated as the GinMAC frame F might become too long. However, such scenarios could still be supported using temporal and spatial transmission diversity.

Temporal And Spatial Transmission Diversity: It is possible to duplicate the basic schedule m times within a GinMAC frame if the overall delay goal permits. Nodes in the deployment are then able to join $m+1$ topologies in which each of them adheres to the set topology envelope. When a node transmits a message, it sends a copy of the message in each of the $m+1$ topologies. The concurrent topologies should be selected such that they do not use the same links whenever possible. The assumption is that copies of the same message use disjoint paths and are therefore not corrupted by infrequent but long burst errors on one link. The number of necessary additional slots $S_{A,h}$ per node at level h can be calculated as:

$$S_{A,h} = m \cdot S_{B,h} \tag{4}$$

The number m must be selected such that an acceptable high packet delivery rate to the sink and to actuators can be achieved. This temporal and spacial transmission diversity clearly costs much more energy than the temporal transmission diversity.

Unused Slots and Energy Consumption: A GinMAC frame consisting of only basic and additional slots may be shorter than the delay requirements would allow ($F < min\{D_S, D_A\}$). In this case, it is useful to add the so-called *unused slots* S_U after the basic and additional slots such that $F = min\{D_S, D_A\}$. A node turns the transceiver off in these unused slots, and thus the energy consumption of a node is improved.

The energy consumption of a node operating GinMAC can be calculated before network deployment. This calculation is useful in industrial deployments where maintenance schedules must be predictable. For each node position in the topology envelope, the worst-case and best-case energy consumption can be determined. The worst-case is incurred if nodes use all assigned slots. In contrast, the best-case is incurred when nodes do not have to forward sensor data and only maintenance messages are transmitted.

As GinMAC is a TDMA protocol, time synchronization is necessary. For this purpose, a node listens every k frames in the first slot that its parent node transmits data upstream. Thus, all nodes synchronize their time with the sink. Every node must always transmit (a packet without payload if no data is available) in the first slot used for upstream data. The packet header contains information on how many packets the sender has to transmit in the current GinMAC frame. Upon receiving the first packet, the receiver knows how many consecutive slots need to be activated for packet reception. If a node does not receive a packet in this first slot, a packet loss is assumed. The node then will listen in the next receive slot. A node might need to use provisioned additional slots to receive all messages. Moreover, a node must transmit in the first slot used to send actuator data to each child node if actuators are located downstream. Again, if a child does not receive data, it will assume a loss and listen in following slots for a retransmission. However, even if no actuators are located downstream, a node still needs to listen in the first downstream slot in case a parent node has to forward non time-critical control messages. If these control messages are lost, they will not be retransmitted within the same GinMAC frame.

Best-Case Energy Consumption: We assume that a transceiver requires the same power p for transmission, reception and idle listening. The power also depends on the task carried out that determines how long a transceiver is active within a slot. The time t the transceiver is active determines the energy consumption $e = p \cdot t$ in a slot. It is assumed that within time t_l the transceiver can determine that no transmission occurs; an acknowledged transmission of a packet without payload requires t_e ; the acknowledged transmission of a full packet requires t_f with $t_l < t_e < t_f$ [1]. The best-case energy consumption E_h^{best} of a node at level h in the topology that does not have to forward actuator data and uses either only basic slots or Temporal Transmission Diversity is obtained as:

$$E_h^{best} = (t_l + (\frac{1}{k} + 1 + O_{h+1}) \cdot t_e) \cdot p \qquad \forall h > 0 \tag{5}$$

In the case that both Temporal and Spacial Transmission Diversity is used, the energy consumption baseline given by E_h^{best} must be multiplied by a factor of $m+1$.

Worst-Case Energy Consumption: If data is transmitted in all available slots, the worst-case energy consumption E_h^{worst} of a node at level h in the topology is incurred as:

$$E_h^{worst} = (\frac{1}{k} + 2 \cdot (S_{B,h} + S_{A,h}) - 1) \cdot t_f \cdot p \qquad \forall h > 0 \tag{6}$$

The duty cycle δ_h of a node at level h is defined as the relation of transceiver on time to total time and can be calculated as $\delta_h = E_h/(F \cdot p)$. For example, assume retransmission slots are not needed, and thus $S_{A,h} = 0$. The energy consumption of a node at level 1 in Figure 2 is $E_1^{best} = (t_l + (\frac{1}{k} + 4) \cdot t_e) \cdot p$ and $E_1^{worst} = (\frac{1}{k} + 23) \cdot t_f \cdot p$. If a CC2420 transceiver, a slot size of $10ms$, a frame length of $F = 1s$ and $k = 100$ are assumed, the duty cycles $\delta_1^{best} = 0.45\%$ and $\delta_1^{worst} = 11.04\%$ can be achieved.

3.3 Topology Control

A node added to the network must determine in which slots it must become active before it can transmit or receive data. The steps used to achieve this are described below.

[1] For example, a CC2420 transceiver requires approximately the same power for transmission, reception and listening. It also uses task times of $t_l = 0.128ms$, $t_e = 1.088ms$ and $t_f = 4.800ms$.

After a node is switched on, it must first obtain time synchronization with the network. Both control and data messages transmitted in the network can be used to obtain this time synchronization. The node continuously listens to overhear a packet from the sink or a node that is already operating in the network. After overhearing one message, the node knows when the GinMAC frame starts as each message carries information about the slot in which it was transmitted.

As a next step, the node must find its position in the topology which must stay within the defined topology envelope. For this purpose, the new node listens for packets in all slots. Transmitted data packets use a header field in which a node that is already a member of the network advertises potentially available positions. For example, a node at position N-1-1-0 shown in Figure 2 may advertise in a header of a data packet traveling to its parent N-1-0-0 that position N-1-1-1 is available. The new node can claim an advertised position by transmitting a data packet in the slot allocated by the advertising node for a potential child node. If an acknowledgement for this transmission is received, the new node has successfully claimed this position in the topology.

A node may be configured with a list of valid nodes that it is allowed to attach to. This might be necessary to ensure that a node will only attempt to join the network using known good links as determined by measurements before the deployment. If a node observes during operation that a link does not fulfill the criteria of a good link, it may decide to attach to the network using a different link. In such a case, a node changing position in the topology must inform its potential child nodes of this event.

If a node looses connectivity to its parent node (determined by an unsuccessful data transmission within one GinMAC frame), it will fall back into the previously described pattern where a node listens on all slots to find a valid attachment point.

4 GinMAC Evaluation

An evaluation is carried out to verify if GinMAC can fulfill the application requirements detailed in Section 2. The protocol is implemented on TinyOS 2.0.2 for TelosB nodes.

4.1 Setup

Ideally, the evaluation should be carried out in an industrial environment such as the GALP oil refinery. However, due to health and safety regulations and possible interference with production, we were not yet able to carry out long-term in situ experiments. Thus, the evaluation testbed is deployed in a corridor of our office building. Nodes are placed on top of metal door frames in a corridor, and consequently communication links can have relatively high loss rates (up to 20%). Such link characteristics are present in the envisioned target scenario where metal pipework obstructs communication paths.

The target platform uses a CC2420 transceiver, and a slot length of $10ms$ is selected. This slot size provides enough time to process and transmit a packet of a maximum payload size and to acknowledge this transmission using hardware acknowledgements. All packets in the experiment have a size of $44bytes$, and thus transmission, reception and listening task times are $t_l = 0.128ms$, $t_e = t_f = 1.952ms$, respectively. Time synchronization is obtained every $k = 10$ frames. The aim is to support a delay bound of $D_S = 1s$ and traffic rates of up to $f = 1Hz$. Nodes are forced into a static binary tree topology with depth $H = 3$ containing $N = 15$ nodes that use links defined as "good links" and by $B_{max} = 1$ and $B_{min} = 1$. The worst-case link reliability is found via pre-deployment measurements. The automatic topology formation, which is described in the previous section, is not used in this evaluation. In the experiments, three different configurations are evaluated: *(Config A)* basic slots only, *(Config B)* basic and additional slots with temporal transmission diversity, and *(Config C)* basic and additional slots with both temporal and spacial transmission diversity.

(Config A) For this configuration, the number of basic slots $S_B = 34$ with $S_B^{up} = 34$ and $S_B^{down} = 0$ is used as the network for evaluation does not contain actuators. The system is provisioned such that a delay bound of $D_S = 1s$ can be provided. However, no mechanisms are in place to handle packet losses. $S_U = 66$ unused slots are added to obtain a GinMAC frame size of $F = 100 slots$, which translates with a slot size of $10ms$ to the required delay bound D_S. In an idle network where there is no data traffic, a leaf node will have a best-case duty cycle of 0.21%. In contrast, under full traffic load where all nodes send a data packet every GinMAC frame, a node directly under the sink will incur a worst-case duty cycle of 2.56%.

(Config B) In this configuration $S_B = 34$ basic slots and $S_A = 34$ additional slots are used to deal with the worst-case channel characteristic given by $B_{max} = 1$ and $B_{min} = 1$. If a transmission fails, additional slots are used for retransmission. $S_U = 32$ unused slots are added to bring the GinMAC frame size to $F = 100 slots$ to provide a delay bound of $D_S = 1s$. In an idle network, a leaf node will have a best-case duty cycle of 0.21%. If under full traffic load, a node directly under the sink will incur a worst-case energy consumption of 5.09%.

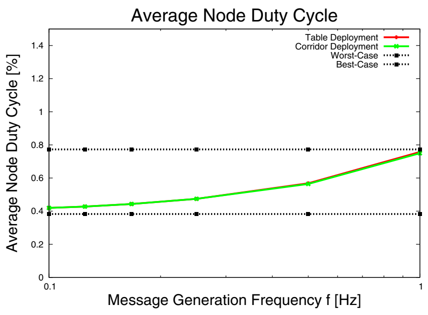
(Config C) In this configuration $S_B = 34$ basic slots and $S_A = 34$ additional slots are used again. The additional slots are used to implement a second disjoint topology. Nodes transmit data using both binary tree topologies to compensate for losses. $S_U = 32$ unused slots are added to bring the GinMAC frame size to $F = 100 slots$ to provide a delay bound of $D_S = 1s$. In an idle network, a leaf node will have a best-case duty cycle of 0.41%. If under full traffic load, a node directly under the sink will incur a worst-case duty cycle of 5.09%.

In all experiments, nodes emit data packets at the traffic rate f where $(0.1Hz \leq f \leq 1Hz)$. The achieved duty cycle δ of all nodes and the message delivery reliability r at the sink are recorded. An experiment lasts for $15 minutes$ for each traffic rate. In addition, five experiments are repeated for each traffic rate to find an average of the energy consumption and delivery reliability measurements.

4.2 Results

Configuration A: Before evaluating the first configuration in the corridor deployment, a separate experiment setup on a table is carried out. The aim of this initial table top evaluation is to determine how the protocol performs in a setting with negligible link errors. In the table top deployment, all 15 nodes are placed in a grid with a spacing of $30cm$, and the configuration similar to that in the corridor deployment is applied.

Figure 4 presents the evaluation results for both table top and corridor deployments. These results are the average node duty cycle $\overline{\delta}$ of all nodes, the average worst-case and average best-case duty cycles of all nodes, and the achieved delivery reliability r.



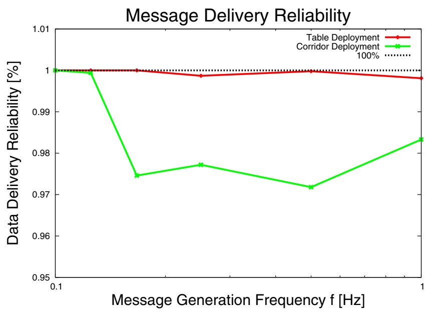


Fig. 4. Average node duty cycle $\overline{\delta}$ and reliability r using configuration A

The achieved duty cycles in both deployments are close to the theoretical best-case for low traffic loads and close to the theoretical worst-case for the maximum traffic rate of $f = 1Hz$. In the table top deployment, delivery reliability r is close to the desired reliability of 100% as transmission errors are rare. However, in the more realistic corridor deployment reliability can drop as low as 97.2% (one leaf node was found to have a high loss rate of 20%), resulting in an unacceptable performance for the applications described in Section 2. Nodes achieve slightly lower duty cycles in the corridor deployment as less data, due to transmission losses, is transported to the sink.

Configuration B: Figure 5 shows the evaluation results for the deployment of configuration B. The delivery reliability is 100% which is desired for the target applications. Thus, the temporal transmission diversity in configuration B is able to compensate link errors seen in the configuration A. Although a price in terms of energy has to be paid to achieve improved reliability, in this case such cost was negligible as only a small number of retransmission slots were required over the duration of the experiment.

Configuration C: Figure 5 depicts the evaluation results for the deployment of configuration C. The delivery reliability is 100% similar to that in configuration B, meeting the requirement of our target applications. However, this method is more costly in terms of energy than configuration B. The increase in energy consumption compared to configuration A is at most 0.53%.

4.3 Findings

The evaluation shows that i) GinMAC can support the target application scenario and ii) the reliability control mechanisms as provided by GinMAC are essential to achieve the required high delivery reliability.

Under the highest traffic load of $f = 1Hz$, the protocol can deliver 100% of data in time with a *node duty cycle* of at most 2.48%, as observed by nodes at level 1 in configuration B. The *average node duty cycle* $\overline{\delta}$ of all nodes in this test scenario is only 0.76%. In an idle network a *node duty cycle* of at most 0.62% is achieved by nodes at level 1, while its *average node duty cycle* $\overline{\delta}$ of all nodes is as little as 0.38%. Common MAC protocols aim for a duty cycle of approximately 2%, which increases with traffic load. Our evaluation illustrates that GinMAC matches this aim at a comparable duty cycle of 2.48% under high traffic load while providing timely and reliable data delivery.

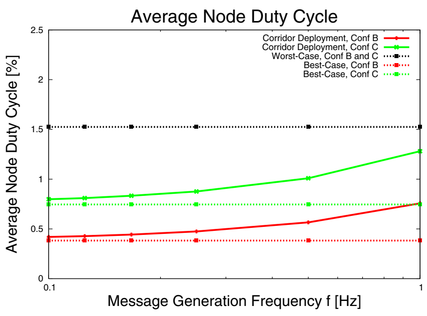


Fig. 5. Average node duty cycle $\overline{\delta}$ using configurations B and C

The presented implementation would allow for a number of optimizations. For example, sensor readings have only a size of a few bytes, but our $10ms$ slots can accommodate much larger packet sizes. Thus, a number of data readings could be transmitted within one slot, which would reduce the number of slots per level. Such an optimization would enable us to conserve more energy or to achieve much tighter delay bounds.

5 Related Work

The WSN research community has to date produced a number of solutions addressing timely data delivery in wireless sensor networks [4]. However, most of these proposals do not completely match the requirements outlined in Section 2. Recent work highlights their shortcomings and points out that more research in this domain is required [2].

Closest to the presented work is the WirelessHART [5] protocol, which is specified by the HART communication foundation. WirelessHART is designed to support industrial process and automation applications. In addition, WirelessHART uses at its core a synchronous MAC protocol called TSMP [6], which combines TDMA and Frequency Division Multiple Access (FDMA). A central entity called Network Manager is used to assign collision free transmission slots and to select redundant routing paths through a mesh network. Thus, the protocol guarantees an upper delay bound while ensuring high transport reliability. Our protocol uses off-line dimensioning to circumvent the complexity and communication overheads that are introduced by this Network Manager.

Prabh [7] specifies a TDMA-based MAC protocol for constructing a network that is dimensioned using scheduling theory. The protocol assumes that a network layout is in a hexagonal shape and that only neighboring nodes in the topology interfere. Based on these assumptions, a carefully designed schedule is devised to achieve the minimum possible bound on message transfer delay. GinMAC is more flexible in terms of topology, and Prabh' s methods are evaluated using only simulations.

Dwarf [8] uses unicast-based partial flooding which limits the degree of transmission redundancy to preserve energy while maintaining reliability. In contrast to our work, all nodes are categorized into rings based on their distance to the nearest sink. The protocol selects a fixed number of forwarding neighbors according to their ring level and wakeup times to decrease an end-to-end delay. When retransmission is needed, a packet is resent to a different forwarding neighbor.

6 Conclusion

The paper details GinMAC that achieves time-critical data delivery in WSNs with extremely low energy expenditure. Hence, the protocol can support industrial process automation and control applications required in the outlined GALP case study. Our evaluation shows that the reliability control mechanisms are practical and essential to deliver the high reliability requirement. In particular, these mechanisms of GinMAC allow us to balance delay, reliability and energy consumption requirements before network deployment. Such achievement is needed for carefully planned industrial control networks. Our next step is to evaluate GinMAC in a deployment at the GALP refinery.

Acknowledgement

This work has been partially supported by the European Commission under the FP7 contract FP7-ICT-224282 (GINSENG).

References

1. Lian, F.L., Moyne, J., Tilbury, D.: Network Design Consideration for Distributed Control Systems. IEEE Trans. Control Syst. Technol. 10, 297–307 (2002)
2. Willig, A.: Recent and Emerging Topics in Wireless Industrial Communications: A Selection. IEEE Trans. Ind. Informat. 4, 102–124 (2008)
3. Munir, S., Lin, S., Hoque, E., Nirjon, S., Stankovic, J., Whitehouse, K.: Addressing Burstiness for Reliable Communication and Latency Bound Generation in Wireless Sensor Networks. In: Proc. 9th Int. Conf. Information Processing in Sensor Networks, Sweden (April 2010)
4. Stankovic, J., Abdelzaher, T., Lu, C., Sha, L., Hou, J.: Real-time communication and coordination in embedded sensor networks. Proc. IEEE 91, 1002–1022 (2003)
5. HART Communication Foundation, WirelessHART Data Sheet (April 2010), http://www.hartcomm.org/
6. Pister, K.S.J., Doherty, L.: TSMP: time synchronized mesh protocol. In: Proc. IASTED Symp. Parallel and Distributed Computing and Systems, Orlando, FL, USA (2008)
7. Shashi Prabh, K.: Real-Time Wireless Sensor Networks. Ph.D. Thesis, Department of Computer Science, University of Virginia, Charlottesville, VA, USA (2007)
8. Strasser, M., Meier, A., Langendoen, K., Blum, P.: Dwarf: Delay-aWAre Robust Forwarding for Energy-Constrained Wireless Sensor Networks. In: Proc. 3rd IEEE Int. Conf. Distributed Computing in Sensor Systems, Santa Fe, NM, USA, pp. 64–81 (2007)

MetroTrack: Predictive Tracking of Mobile Events Using Mobile Phones

Gahng-Seop Ahn[1], Mirco Musolesi[2], Hong Lu[3],
Reza Olfati-Saber[3], and Andrew T. Campbell[3]

[1] The City University of New York, USA
gahn@ccny.cuny.edu
[2] University of St. Andrews, United Kingdom
[3] Dartmouth College, Hanover, NH, USA

Abstract. We propose to use mobile phones carried by people in their everyday lives as mobile sensors to track mobile events. We argue that sensor-enabled mobile phones are best suited to deliver sensing services (e.g., tracking in urban areas) than more traditional solutions, such as static sensor networks, which are limited in scale, performance, and cost. There are a number of challenges in developing a mobile event tracking system using mobile phones. First, mobile sensors need to be tasked before sensing can begin, and only those mobile sensors near the target event should be tasked for the system to scale effectively. Second, there is no guarantee of a sufficient density of mobile sensors around any given event of interest because the mobility of people is uncontrolled. This results in time-varying sensor coverage and disruptive tracking of events, i.e., targets will be lost and must be efficiently recovered. To address these challenges, we propose *MetroTrack*, a mobile-event tracking system based on off-the-shelf mobile phones. MetroTrack is capable of tracking mobile targets through collaboration among local sensing devices that track and predict the future location of a target using a distributed Kalman-Consensus filtering algorithm. We present a proof-of-concept implementation of MetroTrack using Nokia N80 and N95 phones. Large scale simulation results indicate that MetroTrack prolongs the tracking duration in the presence of varying mobile sensor density.

1 Introduction

Urban sensing and tracking [1,5] is an emerging area of interest that presents a new set of challenges for traditional applications such as tracking noise, pollutants, objects (e.g., based on radio signatures using RFID tags), people, cars, or as recently discussed in the literature and popular press, weapons of mass destruction [16]. Traditional tracking solutions [4,7] are based on the deployment of static sensor networks. Building sensor networks for urban environments requires careful planning and deployment of possibly a very large number of sensors capable of offering sufficient coverage density for event detection and tracking. Unless the network provides complete coverage, it must be determined in advance where the network should be deployed. However, it is challenging to determine where the network should be deployed because events are unpredictable in time and space. We believe the use of static networks across urban areas has significant cost, scaling, coverage, and performance issues that will limit their deployment.

An alternative design of such a sensor system, which we propose in this paper, is to use people's mobile phones as mobile sensors to track mobile events. Increasingly, mobile phones are becoming more computation capable and embed sensors and communication support. Therefore, making a sensor network based on mobile phones is becoming more of a reality. For example, many high-end mobile phones, such as Nokia N95 phones, include a number of different radio technologies (e.g., multiple cellular radios, WiFi, and Bluetooth), and sensors (e.g., accelerometer, microphone, camera, and GPS) that are programmable. We imagine that micro-electro-mechanical systems (MEMS) technology will allow for the integration of more specialized sensors (e.g., pollution/air quality sensor, bio sensor, and chemical sensor) in the future. In our design, we assume that we can exploit the mobile phones belonging to people going about their daily lives or defined groups (e.g., federal employees, transit workers, police). Ultimately, the more people who opt in to being a part of the sensor network, the better the density and sensing coverage will be and the more effective urban sensing system will become in delivering services.

There are several important challenges in building a mobile event tracking system using mobile sensors. First, mobile sensors must be tasked before sensing [4]. Another issue that complicates the design of the system is that the mobility of mobile phones (therefore, the mobile sensors) is uncontrolled. This work diverges from mobile sensing systems that use the controlled mobility of a device (e.g., a robot) as part of the overall sensor system design. In such cases, the system can be optimized to drive the mobility of the sensors in response to detected events [11]. Due to the uncontrolled mobility of the mobile sensors, there is no guarantee that there will always be high enough density of mobile sensors around any given event of interest. The density changes over time so that sometimes there is a sufficient number of devices around the event to be tracked, and at other times, there is limited device density. One can think of this as dynamic sensor network coverage. The event tracking process has to be designed assuming that the process of tracking will be disrupted periodically in response to dynamic density and coverage conditions. Thus, a fundamental problem is how to recover a target when the system loses track of the target due to changing coverage.

In this paper, we propose *MetroTrack*, a system capable of tracking mobile events using off-the-shelf mobile phones. MetroTrack is predicated on the fact that a target will be lost during the tracking process, and thus it takes compensatory action to recover the target, allowing the tracking process to continue. In this sense, MetroTrack is designed to be responsive to the changing density of mobile phones and the changing sensor network coverage. The MetroTrack system is capable of tasking mobile sensors around a target event of interest and recovering lost targets by tasking other mobile sensors in close proximity of the lost target based on a prediction of its future location.

MetroTrack is based on two algorithms, namely *information-driven tasking* and *prediction-based recovery*. The tasking is information-driven because each sensor node independently determines whether to forward the tracking task to its neighbors or not, according to its local sensor state information. If the sensor readings meet the criteria of the event being tracked, then the sensor node forwards the task to its neighbors, informing them it detected the event.

The recovery is based on a prediction algorithm that estimates the lost target and its margin of error. MetroTrack uses a geocast approach similar to the algorithms in [12,8] to forward the task to the sensors in the projected area of the target. In our prior work, Olfati-Saber [15] presented the Distributed Kalman-Consensus filter (DKF) that defined the theoretical foundation of distributed tracking of mobile events. In this paper, we extend this work and importantly implement it in an experimental mobile sensing network. We adapt the DKF for the prediction of the projected area of the target.

MetroTrack does not have to rely on a central entity (i.e., a tracking leader) because MetroTrack tracks events based on local state and interactions between mobile phones in the vicinity of a target. Therefore, MetroTrack is simple, flexible, robust, and easy to deploy. However, we do not rule out the potential help from infrastructure. Also, mobile phones occasionally interact with the back-end servers using cellular or infrastructure-based Wi-Fi connectivity for initial tasking purposes or to inform the back-end of the targets progress. In this paper, we focus on the interaction between mobiles and reserve the issues of the interaction with the back-end servers as future work.

Also, we do not discuss what would provide the incentive for more people to opt in (even if we believe mechanisms devised for peer-to-peer systems can be exploited [13,17]), nor do we discuss the important privacy, trust, and security issues that predicate the wide-scale adoption of these ideas. Rather, we leave those issues for future work and focus on the proof of concept and evaluation of a system that is capable of tracking mobile events using mobile phones. To the best of our knowledge, this is the first sensor-based tracking system of mobile events using mobile phones.

The paper is organized as follows. Section 2 describes the information-driven tasking and the prediction-based recovery of MetroTrack. In Section 3, we present the mathematical formulation of the prediction algorithm that is the basis for the prediction-based recovery. In Section 4, we discuss the implementation and the performance evaluation of MetroTrack. A proof-of-concept prototype of MetroTrack is implemented using Nokia N80 and N95 phones to show that MetroTrack can effectively track a mobile noise source in an outdoor urban environment. Following this, in Section 5, we address the large-scale design space of MetroTrack which cannot be analyzed from a small-scale testbed deployment. Section 6 presents some concluding remarks.

2 MetroTrack Design

2.1 Information-Driven Tasking

The *tracking initiation* can be done in two ways, i.e., user initiation or sentry sensor [4] initiation. A user can request to track an event described by certain attributes when the target event is encountered. Another way is to rely on sentry nodes to detect the event to be tracked. The sentry nodes can be selected from mobile nodes that have enough power to periodically turn on their sensors and start sampling. When one of the sentry nodes detects an event that matches the pre-defined event description, the node initiates the tracking procedure. The device associated with the requesting user or first sentry node that has detected the event becomes an initiator.

The *tasking* is a distributed process. Each neighboring sensor node that receives the task message performs sensing. The sensor node does not forward the task message to its neighbors unless it detects the event. The task message is forwarded by the sensors that are tasked and have detected the event. Hence, the nodes in close proximity to the event are tasked and the size of the tasked region is one hop wider than the event sensing range. As a result, the sensors just outside the event sensing range are already tasked and ready to detect the event wherever it moves. Each sensor node locally determines whether it has detected the event by comparing the sensor reading and the description of the event in the task message. As discussed earlier, the description of the event includes the modality of the sensors that can detect the event and the methodology by which the event can be detected (such as a threshold value). If the modality of the sensor node matches one of the modalities specified in the task message (i.e., the device is able to sense the event), then the sensor node starts the sampling process.

The responsibilities of the sensor that detects the event are as follows. The sensor should keep sensing the event using a high sampling rate and report the data to the back-end servers. In addition, the sensor should periodically forward the task message to its neighboring sensor nodes. The sensors that are tasked with one of the task messages containing the same event identifier form a tracking group. We note that this algorithm is not based on the election of a leader. Maintaining a leader for a group requires overhead. In addition, the failure of the leader affects the overall operation of the tracking system. MetroTrack can maintain the group and task the sensors to track the target without the need of a leader.

2.2 Prediction-Based Recovery

This section describes the prediction-based recovery. First, the *recovery initiation* is as follows. The task of tracking the event is distributed among multiple mobile sensors. If a sensor is not detecting the event, this is not considered sufficient to infer that the target is completely lost since other sensors may still be sensing the event. In MetroTrack, a mobile sensor listens to other mobile sensors to minimize the false positives of such decisions. A sensor that has detected the event previously but currently is not detecting the event listens to the task messages forwarded from its neighboring nodes. If none of the neighboring nodes is forwarding the task message, the device infers that the target is lost. Assuming that the speed of the target is comparable to that of a tracking node and the sampling rate of sensors is high enough to detect the event, the overhearing will prevent false positives. However, there might still be false positives if the density of sensors is not sufficient. If a sensor makes a wrong decision, each node will forward an unnecessary number of task requests. However, the penalty is bounded by limiting the duration of the recovery process. In addition, MetroTrack performs suppression to explicitly stop the sensors from forwarding unnecessary messages. When one of the sensors declares that the target is lost, as described above, then the sensor initiates the recovery process by broadcasting a recovery message.

The *recovery process* is based on the estimation of the location of the lost target and the error margin associated to the prediction. The recovery message contains the information about the lost target. MetroTrack adopts a geocast scheme similar to the algorithms in [12,8] to forward the recovery message to the sensors in the projected area in which the target will likely move. The sensors that receive the recovery message attempt to detect the target. If one of the sensors receiving the recovery message detects the target, then the recovery process is complete. The sensor that recovered the target broadcasts a task message, which resumes the information-driven tasking part of the protocol. All the hosts in the recovery area are in the recovery state. We considered a projected circular area. The center of the projected area is the predicted target location and the radius is the error margin of the prediction. The calculation of this area is based on the Kalman filter forecasting techniques, as described in Section 3. MetroTrack calculates the radius R of the recovery area as:

$$R = R_p + R_s + R_c \tag{1}$$

where R_p corresponds to the error margin associated to the prediction (see Equation 8 in Section 3). Our goal is to task all the sensors that are likely to be in contact with the target inside the projected region so we add the sensing range (R_s) to this radius. Finally, we also add the communication range of the devices (R_c) in order to be able to have the nodes that are at a one-hop distance from those at the border of the area with radius R_p+R_s in recovery state. These nodes are likely to enter the area and are particularly useful in spreading the recovery messages in the case of sparse network topologies. We note that the disk-shaped model is an approximated conceptual model that, in a real deployment, is influenced by the GPS errors for localization and by non uniform radio propagation and interferences. A node that has received the recovery message stays in recovery state until the node moves outside the recovery area or the recovery process timer expires. A timeout is specified to limit the duration of the recovery process. If the target is not recovered after the expiration of the timer, MetroTrack stops tracking the target. The nodes in recovery state periodically broadcast the recovery message to their instant (one-hop) neighbors so that new nodes that move into the recovery area can receive the recovery message.

It may happen that some sensors can be still in the recovery state while other sensors have already recovered the target and started to track it. It may also happen that the target event disappears (e.g., a sound source that is suddenly silent). MetroTrack addresses this problem using two mechanisms. First, it limits the duration of the recovery process and the spatial dissemination of the recovery messages. Second, MetroTrack performs a *suppression process* to reduce unnecessary overhead. Every node that recovers the target or receives a task message broadcasts a suppression message that is disseminated among the devices in the recovery area. Every node that receives the suppression message inside the recovery area re-broadcasts the message, or, if the node is in recovery state, it stops the recovery process and stops broadcasting the recovery message.

3 Prediction Algorithm

3.1 Prediction Model

In this section, we provide an overview of the prediction model in order to fully understand the collaborative prediction protocol used for the recovery process. We define a generic model for predicting the movement of a target in geographical space based on the Constant Velocity model [3], which is widely used in mobile tracking. Despite of the term 'constant velocity', the Constant Velocity model represents a moving target with dynamically changing velocity with certain variance. We consider a moving target with position $q \in \Re^2$ and a velocity $p \in \Re^2$. The one-step predictor is defined as follows:

$$\hat{x}(k+1) = A\bar{x}(k) + Bw(k) \tag{2}$$

where $x(k) = [q_1(k), p_1(k), q_2(k), p_2(k)]$ denotes the state of the target at time k. $\bar{x}(k)$ indicates the *prior state estimate* at step k given the knowledge of the movement under observation, whereas $\hat{x}$ indicates the *state estimate* of the same process at time $k+1$. q_1 and p_1 are the position and the speed on the x-axis and q_2 and p_2 are the position and speed on the y-axis, respectively. $w(k)$ is a zero-mean Gaussian noise denoted by $N(0,1)$. The prior estimate is the information stored in the phones and periodically exchanged among the phones that are in reach. The matrix A and B are defined as follows:

$$A = \begin{pmatrix} 1 & \epsilon & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \epsilon \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad B = I_2 \otimes G, \quad \textit{with} \quad G = \begin{pmatrix} \epsilon^2\sigma_0/2 \\ \epsilon\sigma_0 \end{pmatrix}$$

where ϵ is the interval of steps and $\otimes$ denotes the Kronecker product of matrices. The prediction for the instant $k+2$ is defined as follows:

$$\hat{x}(k+2) = A^2\bar{x}(k) + ABw(k) + Bw(k+1) \tag{3}$$

The generic prediction for the instant $k+m$ is defined as:

$$\hat{x}(k+m) = A^m\bar{x}(k) + \sum_{j=0}^{m-1} A^j Bw(k+m-1-j) \tag{4}$$

The meaning of the symbols $\hat{x}$ and $\bar{x}$ is the same of the $k+1$ case. This equation can be rewritten as:

$$\hat{x}(k+m) = A^m\bar{x}(k) + v(k) \tag{5}$$

where $v(k)$ is the noise associated to the $k + m$ prediction defined as:

$$v(k) = \sum_{j=0}^{m-1} A^j Bw(k + m - 1 - j) \tag{6}$$

The variance of $v(k)$ is

$$R_v = \begin{pmatrix} \sigma^2_{v_{q_1}} & 0 & 0 & 0 \\ 0 & \sigma^2_{v_{p_1}} & 0 & 0 \\ 0 & 0 & \sigma^2_{v_{q_2}} & 0 \\ 0 & 0 & 0 & \sigma^2_{v_{p_2}} \end{pmatrix} = [\sum_{j=0}^{m-1} A^j BB^T (A^j)^T] R_w \tag{7}$$

where $R_w = I_4$. Therefore, the center of the recovery region is $(\hat{q}_{(k+m)_1}, \hat{q}_{(k+m)_2})$. We consider a radius for the recovery area equal to:

$$r = max[2\sigma_{v_{q_1}}, 2\sigma_{v_{q_2}}] \tag{8}$$

The value of r is chosen in order to obtain a 95% confidence interval for the projected recovery area. In other words, we can assume that the target will be located in the recovery area with approximately 95% probability.

3.2 Distributed Kalman-Consensus Filter

In our prior work, Olfati-Saber [15] presented the Distributed Kalman Consensus Filter that defined the theoretical foundation of distributed tracking of mobile events. Algorithm 1 is the outcome of [15]. We feed our prediction model presented in the previous section to Algorithm 1 to predict the location of the target after it is lost and to calculate the projected area for the recovery process.

Each node i runs the distributed estimation algorithm shown in Algorithm 1. We indicate with z_i the observation performed by each node. N_i indicates the neighbors of node i. The message that is periodically broadcasted contains the following tuple: $msg_i = [u_i, U_i, \hat{x}_i]$. The local aggregation and calculation is described in step 3, whereas the estimation of the consensus among the neighbors is performed in step 4. The equations of the update of the filter are presented in step 5.

The sensing model that we use is the following:

$$z_i(k) = H_i(k)x(k) + v_i(k) \tag{9}$$

where $H_i(k)$ is the observation matrix and $v_i(k)$ is the zero-mean Gaussian noise of the measurements of the ith node with covariance R_i. In our implementation, we assume that the value of the observation matrices $H_i(k)$ is the same for the all nodes over time and it is equal to:

$$H = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix}$$

Algorithm 1. Distributed Kalman Consensus Filter

```
1. Initialization: $P_i = P_0, \bar{x}_i = x(0)$
2. while new data exists do
3.    Locally aggregate data and covariance matrices:
      $J_i = N_i \cup \{i\}$
      $u_j = H_j^T R_j^{-1} z_j,\ \forall j \in J_i,\ y_i = \sum_{j \in J_i} u_j$
      $U_j = H_j^T R_j^{-1} H_j,\ \forall j \in J_i,\ S_i = \sum_{j \in J_i} U_j$
4.    Compute the Kalman-Consensus estimate:
      $M_i = (P_i^{-1} + S_i)^{-1}$
      $\hat{x}_i = \bar{x}_i + M_i(y_i - S_i \bar{x}_i) + \epsilon M_i \sum_{j \in N_i} (\bar{x}_j - \bar{x}_i)$
5.    Update the state of the Kalman-Consensus filter:
      $P_i \leftarrow A M_i A^T + B Q B^T$
      $\bar{x}_i \leftarrow A \hat{x}_i$
6. end while
```

We also assume that the value of R_i is equal to a constant for all the matrices:

$$R_i = \sigma_R^2 I_2 \tag{10}$$

The value of Q is the same for all the devices since it is only dependent on the value of the process under observation that is the same for all the devices (i.e., the position of the moving target):

$$Q = \sigma_0^2 I_4 \tag{11}$$

Finally, P_0 is defined as

$$P_0 = \sigma_R^2 I_4 \tag{12}$$

4 Implementation and Experiment

4.1 Implementation

We built a proof-of-concept, mobile phone-based testbed to evaluate the MetroTrack system. The testbed consists of Nokia N80 and N95 smart phones (shown in Figure 1(a)) running Symbian OS S60. Both of them are equipped with a microphone and a camera that are accessible via software. With respect to network connectivity, they are both equipped with Bluetooth and WiFi interfaces.

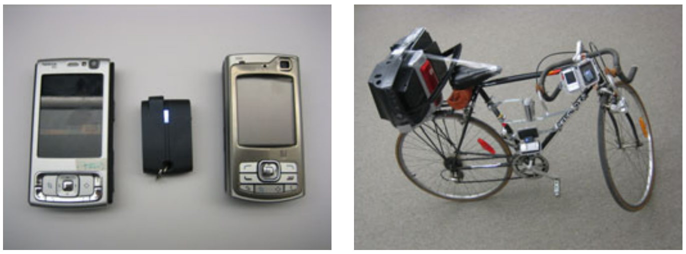

(a) (b)

Fig. 1. (a)From left to right: N95, GPS dongle, N80. (b)The boombox bike.

The N95 phones also feature an integrated GPS and an accelerometer. Since the N80 phones are not equipped with a GPS, we used an external dongle (shown in Figure 1(a)) based on the SiRFstar III chipset connected to the phone via Bluetooth. The devices use GPS information for sound source localization and the recovery process. In our testbed, we used WiFi for local ad hoc communications between mobile phones and used UDP broadcasting. The MetroTrack system is written in PyS60 [14], Nokia's porting of Python 2.2 for Symbian OS S60. Currently, PyS60 is more flexible than the Nokia implementation of J2ME for the N80 and N95 phones with respect to the programming interface for accessing the sensors embedded on the phones. With respect to the Symbian C++ development environment, it provides high-level abstractions that are extremely useful and convenient for the rapid prototyping of applications.

We implement an experimental sound source tracking application interfaced with MetroTrack. The system architecture is illustrated in Figure 2. We record sound samples using the microphone every 2 s. To estimate the distance from the target, we compute the Root Mean Square (RMS) of the average sound signal amplitude. If the calculated RMS value is distinctively greater than the ambient noise level, the sensor determines that the target event is detected and feeds the RMS value to the distance estimation component. An alternative method is *bearing estimation* [6], but it is not applicable to mobile phones due to the requirement of two microphones on one device with known orientation.

We implement two prediction mechanisms, a local Kalman filter (LKF) [10] and a consensus-based distributed Kalman filter (DKF) [15] in order to evaluate the trade-offs between the two. The LKF is simply a special case of the DKF without sharing information among neighboring devices. We implement the DKF, as described in Section 3. With respect to the mathematical model presented in Section 3, for the LKF, we assume that $J_i = \emptyset \cup \{i\}$.

The distance between the sensor and the sound source can be estimated from the RMS value assuming that we know the original volume of the target sound and the pattern of the sound attenuation over distance. The prototype is based on a trilateration, which is a widely used localization scheme in GPS. After estimating its location and distance from the target, each sensor shares this information with its one-hop neighbors for trilateration, which require distances from two reference points for 2-D localization. The target location estimated by

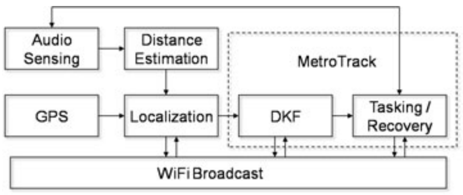


Fig. 2. System Architecture

the sound source localization is fed into the Distributed Kalman filter component as the observation of the node.

4.2 Experiment

We mount a boombox, which plays constant pink noise (i.e., a signal with a frequency spectrum such that the power spectral density is proportional to the reciprocal of the frequency), on the back of a bike (aka boombox bike). We move it at a slow pace along paths around a university campus at approximately walking speed. We set the speaker of the boombox to face down toward the ground (as shown in Figure 1(b)) so that the sound would be reflected and spread omni-directionally in 2-D dimensions.

We set up a tracking testbed composed of two N95 phones and nine N80 phones connected to nine Bluetooth GPS dongles. The sound is sampled by the microphone on each phone for 0.5 s. The sampling is performed every 2 s. The time interval between each sampling is 1.5 s. Because the mobile phones are not always performing the tracking process (i.e., it can be defined as opportunistic), we argue that the maximum achievable sampling rate and minimum transmission interval of the messages should be used. Energy cost is not an issue if the device is not frequently involved in the tracking process. The values of the intervals are those sufficient for both the N80 and N95 phones for the RMS calculation, the distance estimation, the sound source localization, and the Distributed Kalman filter update calculation. We note that in existing tracking systems, the time intervals are much smaller than those used in MetroTrack (i.e., approximately 0.1-0.2 s) [7]. We set the WiFi transmission power to 100 mW. The communication range is between 25 and 30 m.

We perform the sound source tracking experiment evaluating the accuracy of the sound source localization as well as the effectiveness of the MetroTrack tasking and recovery. The GPS trace of the target is shown in Figure 3(a). Each person carries a phone and a Bluetooth dongle. Given the limitation of the number of phones and people, we emulate the density of an urban setting by allowing people to move around within 40 m from the target (i.e., the boombox bike). Given the restriction of being within 40 m from the target, each person was allowed to move randomly in and out of the sensing range (approximately 20 m). This mobility setup is sufficient for testing the effectiveness of the tasking process. We emulate the case of losing the target by turning the sound off for 16

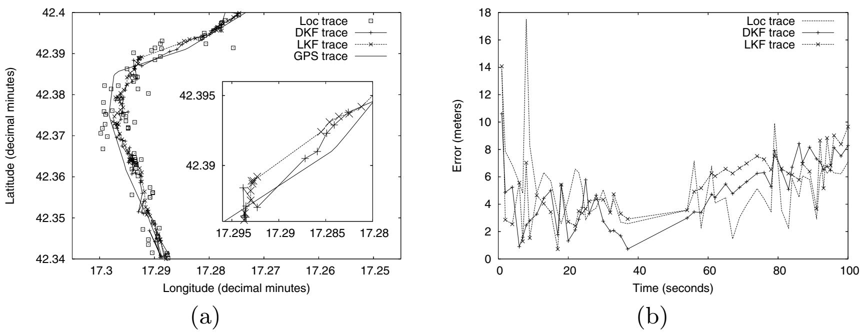


Fig. 3. (a) Trace of target's location. (b) Time trace of the localization error.

s and then turning the sound on again to observe whether the recovery process is working effectively.

The trace of the target measured using the sound source localization scheme is shown in Figure 3(a). (See the curve *Loc trace* in the plot.) As observed in Figure 3(a), the measured location is noisy. The sound source localization error is not only caused by the error of the RMS measurement but also by the error of the GPS positioning estimation of the mobile sensors. Each mobile sensor uses its own GPS receiver, and the accuracy of these receivers varies, even if they are of the same model. Also, some mobile sensors do not have valid GPS readings at all on a cloudy day. We have learned that calibrating the GPS reading among different sensors and checking the integrity of the GPS position of the mobile sensors is a real challenge that needs to be addressed in the future. The inset in Figure 3(a) shows a zoomed section of the gap in the traces related to the recovery phase. For clarity, the localization traces are not shown in the inset.

We also test the LKF and DKF estimations by setting σ_R to 7 m because we learned by trying the experiment several times that the standard deviation of the sound source localization error (σ_R) is approximately 7 m. The trace of the LKF and DKF estimations of the target location is also shown in Figure 3(a). In order to show the correctness of the prediction mechanism, we plot the time trace of the error of the location estimation in Figure 3(b). The target in this figure starts at instant $t = 0$ from the top of the area to the bottom. In Figure 3(b) we show the time interval of the first 100 seconds, including the interval during which the target was lost (i.e., between time $t = 37$ s and $t = 54$ s). We observe that the estimation error of DKF is smaller than the error of the LKF.

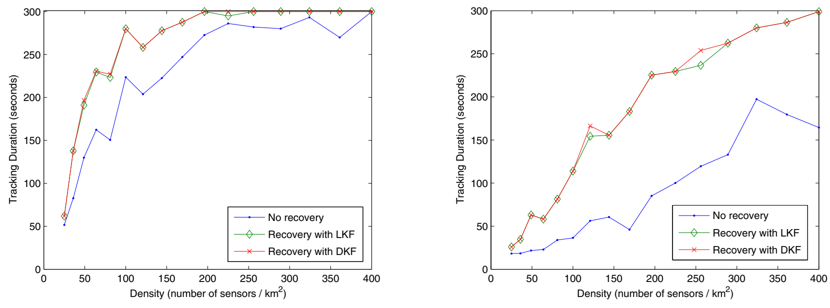
5 Simulation Study

We evaluate the performance of MetroTrack for a number of different deployment scenarios using a time-driven simulator based on MATLAB. The simulation results complement the experimental evaluation by studying issues not easily evaluated in a small-scale testbed, such as scaling and a sensitivity analysis of the system. In this section, we show the tracking duration performance for various sensor densities and sensing ranges.

We run each simulation scenario 20 times with each simulation duration of $300s$. The target event is active from the beginning to the end of every simulation run. The simulation area is a $1000m \times 1000m$ square. We assume an omni-directional radio model with a transmission range of $100m$. The sensing ranges of $100m$ and $50m$ are tested. If the target is within the sensing range of a tasked sensor, the sensor is able to estimate the location of the target. The distribution of the localization error is modeled using a zero-mean Gaussian distribution with standard deviation $\sigma_R = 20m$. Targets characterized by mobility patterns with larger standard deviations are more difficult to track. Every tasked sensor estimates the location of the target once in every sampling interval of $1s$. The timeout value for the recovery process is $20s$.

For the mobility of mobile sensors and the target, we consider the Constant Velocity model [3], which is the underlying model that MetroTrack uses for the Kalman filters, as discussed in Section 3. Initially, mobile sensors are randomly placed according to a uniform distribution on the plane. The standard deviation of the movement dynamics of the target and sensor nodes σ_0 is $0.2m/s$. When a target or sensor node reaches the boundary of the simulation area, it changes its direction toward one of the other sides of the simulation area. We have also simulated two other widely used mobility models, the Random Way-point model [9] and the Manhattan model [2]. The results are basically similar to the simulation study using the Constant Velocity model.

One of the main objectives of MetroTrack is to track the target for as long as possible without losing it. Therefore, the duration of tracking is one of the main performance metrics. Figure 4 shows the tracking duration with varying densities and sensing ranges of mobile sensors. We measure the duration of tracking when MetroTrack performs the information-driven tasking but it does not perform the prediction-based recovery (no recovery). We then measure the duration of tracking when MetroTrack performs the prediction-based recovery as well. We compare the tracking duration when MetroTrack uses the Distributed Kalman filter (Recovery with DKF) and when it uses the Local Kalman filter (Recovery with LKF). The x-axis is the density of sensors, and the y-axis is the duration of tracking. We run the simulation for 300 s. The tracking starts from the beginning of the simulation. The target is lost before the simulation ends. As observed in Figure 4, the prediction-based recovery prolongs the duration of the tracking. Moreover, the recovery enables the tracking to last until the end of simulation with the densities of greater than 200 sensors or more per km^2 if the sensing range is $100m$. If the sensing range is 50 m as in Figure 4(b), the recovery enables the tracking to last until the end with a density of 400 sensors. The extended duration by the recovery process is longer for the $50m$ sensing range than for the $100m$ sensing range. It is interesting that the recovery processes using both filters do not show much difference in tracking duration, whereas the DKF showed better accuracy in prediction. The forwarding zone is the sum of the radius of the recovery region, the sensing range, and the communication



(a) Sensing range of 100 m. (b) Sensing range of 50 m.

Fig. 4. Tracking duration vs. density of mobile sensors

range, as we explained earlier. The size of the forwarding zone is big enough to absorb the impact of the inaccuracy of the prediction of the LKF.

6 Summary

In this paper, we proposed MetroTrack, the first distributed tracking system that tracks mobile events using off-the-shelf mobile phones. We presented the design and implementation of the system and discussed the mathematical foundations upon which our distributed prediction models are based. We evaluated the system through the deployment of a prototype implementation of the system using Nokia N80 and N95 mobile phones and analyzed the performance of the system for a number of different scenarios through simulation. While the proof-of-concept prototype implementation of MetroTrack focused on tracking a mobile audio source, we believe that the algorithms and techniques discussed in this paper are more broadly applicable to an emerging class of problems related to the efficient tracking of mobile events using off-the-shelf mobile devices such as mobile phones, PDAs, and mobile embedded sensors.

Acknowledgement

This work is supported in part by Intel Corp., Nokia, NSF NCS-0631289, and the Institute for Security Technology Studies (ISTS) at Dartmouth College. ISTS support is provided by the U.S. Department of Homeland Security under award 2006-CS-001-000001, and by award 60NANB6D6130 from the U.S. Department of Commerce. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any funding body.

References

1. Abdelzaher, T., Anokwa, Y., Boda, P., Burke, J., Estrin, D., Guibas, L., Kansal, A., Madden, S., Reich, J.: Mobiscopes for human spaces. IEEE Pervasive Computing 6(2), 20–29 (2007)

2. Bai, F., Sadagopan, N., Helmy, A.: IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks. In: INFOCOM 2003, San Francisco, CA, USA (April 2003)
3. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications and Mobile Computing. Special issue on Mobile Ad Hoc Networking 2(5), 483–502 (2002)
4. Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., Zhao, J.: Habitat Monitoring: Application Driver for Wireless Communications Technology. In: Workshop on Data Communications in Latin America and the Caribbean (April 2001)
5. Cuff, D., Hansen, M., Kang, J.: Urban sensing: Out of the woods. Communications of the ACM 51(3), 24–33 (2008)
6. Girod, L., Lukac, M., Trifa, V., Estrin, D.: The design and implementation of a self-calibrating distributed acoustic sensing platform. In: 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006), pp. 71–84 (2006)
7. He, T., Krishnamurthy, S., Stankovic, J.A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Zhou, G., Hui, J., Krogh, B.: VigilNet: An Integrated Sensor Network System for Energy-Efficent Surveillance. ACM Transactions on Sensor Networks (2004)
8. Huang, Q., Lu, C., Roman, G.-C.: Spatiotemporal Multicast in Sensor Network. In: First ACM Conference on Embedded Networked Sensor Systems, SenSys 2003 (2003)
9. Johnson, D., Maltz, D.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: Mobile Computing, pp. 153–181 (1996)
10. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME Journal of Basic Engineering (March 1960)
11. Kansal, A., Somasundara, A.A., Jea, D.D., Srivastava, M.B., Estrin, D.: Intelligent fluid infrastructure for embedded networks. In: MobiSys 2004, pp. 111–124. ACM, New York (2004)
12. Ko, Y.-B., Vaidya, N.: Geocasting in Mobile Ad Hoc Networks: Location-based Multicast Algorithms. In: Workshop on Mobile Computer Systems and Applications (WMCSA 1999) (February 1999)
13. Lai, K., Feldman, M., Stoica, Chuang, J.: Incentives for cooperation in peer-to-peer networks. In: Workshop on Economics of Peer-to-Peer Systems (2003)
14. Nokia. Python for s60, http://wiki.opensource.nokia.com/projects/PyS60
15. Olfati-Saber, R.: Distributed Kalman Filtering for Sensor Networks. In: 46th IEEE Conference on Decision and Control (December 2007)
16. Purdue University. Cell phone sensors detect radiation to thwart nuclear terrorism, http://news.uns.purdue.edu/x/2008a/080122FischbachNuclear.html
17. Shneidman, J., Parkes, D.C.: Rationality and self-interest in peer to peer networks. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 139–148. Springer, Heidelberg (2003)

Mobile Sensor Network Localization in Harsh Environments

Harsha Chenji and Radu Stoleru

Texas A&M University, College Station TX, USA
{cjh,stoleru}@cse.tamu.edu

Abstract. The node localization problem in mobile sensor networks has recently received significant attention. Particle filters, adapted from robotics, have produced good localization accuracies in conventional settings, but suffer significantly when used in challenging indoor and mobile environments characterized by a high degree of radio irregularity. We propose FuzLoc, a fuzzy logic-based approach for mobile node localization in challenging environments and formulate the localization problem as a fuzzy multilateration problem, with a fuzzy grid-prediction scheme for sparse networks. We demonstrate the performance and feasibility of our localization scheme through extensive simulations and a proof-of-concept implementation on hardware, respectively. Simulation results augmented by data gathered from our 42 node indoor testbed demonstrate improvements in the localization accuracy from 20%-40% when the radio irregularity is high.

1 Introduction

Wireless sensor networks are increasingly a part of the modern landscape. Disciplines as diverse as volcanic eruption prediction [1] and disaster response [2] benefit from the addition of sensing and networking. One common requirement of many wireless sensor network (WSN) systems is localization, where deployed nodes in a network endeavor to discover their positions. The precise location of a pre-eruption tremor or of a patient in distress are two compelling examples of the need for accurate localization.

In some cases, localization is simple. For smaller networks covering small areas, fixed gateway devices and one-hop communications provide enough resolution. Larger networks may be provisioned with location information at the time of deployment [3]. GPS is a viable option for small outdoor deployments where cost and the power budget permit. However, in many common environments, localization is very difficult. GPS-based localization is not viable when the GPS receiver can't see the satellites. Signal strength-based solutions fail when there is a high degree of RF multi-path or interference, like most indoor and urban environments. Mobility in these harsh environments further complicates the problem.

Some solutions for localization in harsh, mobile environments assume availability of high-precision clocks and specialized hardware. [4] and [5] rely on accurate measurement of TDOA and distance traveled. Radio interferometry localizes nodes to within centimeters in [6] in non-multipath environments. All of these solutions rely on stable environments with low multi-path where a measured or sensed range reliably predicts the actual distance between two nodes.

Harsh, mobile environments, however, fail to meet these assumptions. The RSS-distance relationship [7] is inconstant and connectivity may vary dramatically. Assuming a predictable relationship between distance and RSS is problematic due to errors induced by multi-path and fading. Connectivity information gathered by the nodes could actually be misinformation. In many cases, compounding of errors may occur if the localization method relies on previous location estimations.

Fuzzy logic offers an inexpensive and robust way to deal with highly variable RSS measurements made in noisy, uncertain environments. Empirical measurements are used to produce rules that the fuzzy inference system uses to interpret input. The output of this process recovers the actual value compensated for variability in the local environment. We employ this basic technique in two constituent subsystems of FuzLoc - the Fuzzy Non Linear System (FNLS) and the Fuzzy Grid Prediction System (FGPS).

Our contributions include: i) a fuzzy logic-based method of building rule sets that characterizes the local signal environment, ii) a fuzzy non-linear system construct that uses these rules to convert extremely noisy RSS measurements to locations using a mechanism that compensates for inherent error, iii) a fuzzy grid prediction system that optimizes localization under conditions of low connectivity and low anchor density, iv) extensive simulations and comparisons to state of the art algorithms and v) a proof-of-concept implementation on motes.

The paper is organized as follows. Section 2 motivates our work and Section 3 introduces the fuzzy logic-based framework. Section 4 evaluates the proposed localization technique, followed by Conclusions in Section 5.

2 Motivation and Background

Several authors [8,9,10,11] have proposed Monte Carlo-based techniques, frequently used in robotics, for localization in mobile sensor networks. These localization techniques assume that a subset of nodes, called anchors, know their location. Nodes and anchors move randomly in the deployment area. Maximum velocity of a node is bounded but the actual velocity is unknown to nodes or anchors. Anchors periodically broadcast their locations.

This paper is motivated by our interest in a localization technique for a mobile sensor network, deployed in a harsh environment and the set of interesting/ surprising results obtained from simulations of three state of the art localization techniques for mobile sensor networks, namely MCL [8], MSL [9] and OTMCL [11]. Using the simulators developed by the authors of [8,9,11], a scenario assuming highly irregular radio ranges was developed, typical of harsh indoor or extremely obstructed outdoor environments. The irregularity in the radio range is modeled in these simulators as a degree of irregularity (DoI) parameter [8]. The DoI represents the maximum radio range variation per unit degree change in direction.

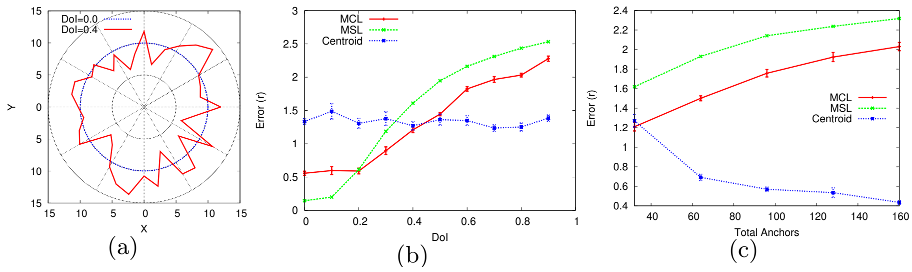


Fig. 1. (a) Radio patterns for two different degrees of radio irregularity (DoI); (b) the effect of DoI on localization error in MSL, MCL and Centroid; and (c) the effect of anchor density on localization error, at DoI=0.4, for MSL, MCL and Centroid

An example is depicted in Figure 1(a). When DoI=0.4 the actual communication range is randomly chosen from [0.6r, 1.4r].

Simulation results, for a network of 320 nodes, 32 anchors deployed in a 500×500 moving at 0.2r (r, the radio range) are shown in Figures 1(b) and 1(c). Figure 1(b) demonstrates that the DoI parameter has a significant negative effect on the localization accuracy. At DoI=0, both MCL, MSL and OTMCL achieve localization errors of 0.2r and 0.5r. With an increase in the DoI to 0.4, their localization error increases 400%. More surprisingly, as depicted in Figure 1(c), at a high DoI value, an increase in the number of anchors has a detrimental effect on localization accuracy. This result is counter-intuitive since access to more anchors implies that nodes have more opportunities to receive accurate location information, as exemplified by the performance of Centroid [12], in the same figure. A similar observation is made in [10] although no further study was performed. Our results suggest that samples get successively polluted with time, since the nodes used for filtering the samples may not be actual neighbors. The number of polluted samples increases with increasing anchor density.

The challenges identified above were partially addressed in recent work in sensor network node localization [13] that makes use of variables typical of range-based localization techniques (e.g., RSSI) to improve the accuracy of range-free techniques. In a similar vein, we propose to formulate the localization problem as a fuzzy inference problem using RSSI in a fuzzy logic-based localization system where the concept of distances used are very loose, such as “High”, “Medium” or “Low”.

2.1 Related Work

Existing work can be classified as range-based or range-free although a few techniques do not fall cleanly into these categories.

Range-based Localization Methods: These methods require an estimate of the distance or angle between two nodes to localize and may operative with both absolute and relative coordinate systems. A frequent requirement is the presence of at least three anchors so that necessary uniqueness and geometric constraints are satisfied. GPS is a familiar range-based method that uses the time of arrival of signals from satellites to obtain a precise location in latitude-longitude format. Some methods use surveying to predetermine RSSI values at any point in the area of deployment. Many solutions use time difference of arrival (TDoA) [4] [5]. For all of these methods, typical drawbacks include additional hardware, higher computational loads, increased node size, higher energy consumption and increased cost. A lighter weight solution leverages existing cellular telephony networks. Fuzzy logic is used to locate cellular phones in a hexagonal grid in a cellular network [14]. It assumes a fixed number of anchors but handles mobility very well. The computation and refining are not suitable for resource-constrained embedded sensors.

Range-free Localization Methods: Hop counting is a technique frequently used in these scenarios. A major drawback for hop-counting is that it fails in networks with irregular topologies such as those with a concave shape. Mobility incurs large overhead since all the hop counts will have to be refreshed frequently. APIT is a similar method which divides the area of deployment into triangles formed by anchors and then estimates the location. It assumes a large anchor density and higher radio ranges for the anchor nodes. An advantage is, however, that extremely low computational resources are needed. Both methods introduce large errors for ad-hoc networks. A hybrid method [13] that uses RSS and connectivity is worthy of note.

2.2 Background

Fuzzy logic has been applied in robot localization [15,16], because it reworks classical set theory and enables it to have non-rigid, or fuzzy, set boundaries. A *fuzzy set*, sometimes called *fuzzy bin*, is defined by an associated function $\mu(x)$, which describes the degree of membership of a crisp number in the set. A *crisp number* can belong to more than one fuzzy set at a given time, with varying degrees of membership. A *fuzzy number* is a special fuzzy bin where the membership is 1 at one and only one point. A fuzzy number represents a multi-valued, imprecise quantity unlike a single valued traditional number. One popular $\mu(x)$ function, depicted in Figure 2(a), is the *triangular membership function*:

$$\mu(x) = \begin{cases} 0 & \text{if } x < a \\ (x-a)/(b-a) & \text{if } a \leq x \leq b \\ (c-x)/(c-b) & \text{if } b \leq x \leq c \\ 0 & \text{if } x > c \end{cases} \tag{1}$$

where (a, b, c) defines a triangular bin. As shown, the $WEAK$ fuzzy set can be represented as (-90,-70,-50) and $MEDIUM$ as (-70,-50,-30). A crisp number, RSSI = -55dBm has a membership of 0.25 in $WEAK$ and 0.75 in $MEDIUM$.

A *fuzzy system* is defined by a set of fuzzy rules which relate linguistic variables in the form of an IF-THEN clause. Typically the IF clause contains the input linguistic variable (e.g., RSSI) and the THEN clause contains the output linguistic variable (e.g., DISTANCE).

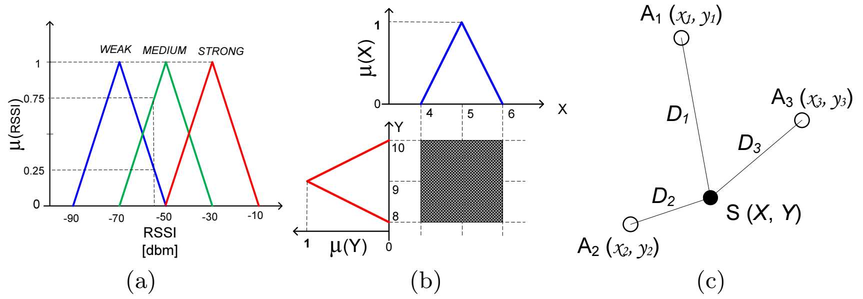


Fig. 2. (a) A triangular membership function $\mu_{(RSSI)}$ consisting of three bins. A fuzzified RSSI value of -55 pertains to two bins, $WEAK$ and $MEDIUM$, with degrees of membership of 0.25 and 0.75, respectively; (b) Representation of a fuzzy location, using two triangular membership functions; and (c) a sensor node S with fuzzy coordinates X and Y, to be located using three anchors positioned at (x_1, y_1), (x_2, y_2) and (x_3, y_3).

3 A Fuzzy Logic-Based Node Localization Framework

The two dimensional location of a node can be represented as a pair (X, Y), where both X and Y are fuzzy numbers, as depicted in Figure 2(b). This section develops the theoretical foundation behind the computation of this fuzzy location, using imprecise and noisy RSSI measurements labeled as “HIGH”, “MEDIUM” or “LOW”.

3.1 Fuzzy Non Linear System (FNLS)

As depicted in Figure 2(c), consider a node s about to be localized, in the vicinity of three anchor nodes A_j ($j = \overline{1,3}$). Each anchor node is equipped with a set of fuzzy rules that map fuzzy RSSI values to fuzzy distance values:

Rule i: **IF** RSSI is $RSSI_i$ **THEN** DIST is $Dist_i$

where $RSSI_i$ and $Dist_i$ are fuzzy linguistic variables (e.g. LOW, $MEDIUM$, $HIGH$) and “is” means “is a member of”.

For a more general case, when the node S is within radio range of n anchors, the node localization problem can be formulated as a fuzzy multilateration problem. The following:

$$\begin{aligned} F_1 &= (X - x_1)^2 + (Y - y_1)^2 - D_1^2 = 0 \\ F_2 &= (X - x_2)^2 + (Y - y_2)^2 - D_2^2 = 0 \\ &\dots \\ F_n &= (X - x_n)^2 + (Y - y_n)^2 - D_n^2 = 0 \end{aligned} \tag{2}$$

defines a non-linear system of equations describing the relation between the locations of the nodes and anchors and the distances among them. The variables

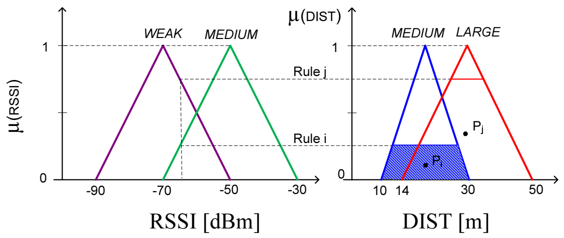


Fig. 3. The fuzzification process for an input RSSI value of -62dB. In this example, the fuzzy rule base maps this value through two rules: "Rule i: IF RSS is WEAK, THEN distance is LARGE" and "Rule j: IF RSS is MEDIUM, THEN distance is MEDIUM".

X, Y and D_k $(k = \overline{1,n})$ are fuzzy numbers, while (x_k, y_k) $(k = \overline{1,n})$ are crisp numbers. The objective is to minimize the mean square error over all equations.

FIS Subsystem. A definition of the process of obtaining the fuzzy distances D_k is needed before solving the system of equations. This process, called fuzzy inference, transforms a crisp RSSI value obtained from a packet sent by a node and received by an anchor into a fuzzy number, i.e., distance D_k between node and anchor. Figure 3 depicts an example for the fuzzifying process. As shown, an RSSI value of -62dBm has different membership values $\mu(RSSI)$ for the fuzzy bins $WEAK$ and $MEDIUM$. The two fuzzy bins, in this example, are mapped by a fuzzy rule base formed by two fuzzy rules: "Rule i" and "Rule j". These two fuzzy rules define the mapping from the RSSI fuzzy sets to the DIST fuzzy sets. As shown in Figure 3, the two fuzzy rules indicate the membership $\mu(DIST)$ in the distance domain. P_i and P_j indicate the center of gravity of the trapezoid formed by the mapping of the RSSI into fuzzy bins $MEDIUM$ and $LARGE$, respectively.

Typically, a single RSSI value matches multiple fuzzy rules. Let's assume that the fuzzy rule base maps an RSSI value to a set of m fuzzy DIST bins. The set of centers of gravity P_l $(l = \overline{1,m})$ is denoted by $P = \{P_1, P_2 \ldots P_m\}$ in Figure 3. The intuition for computing the output as a fuzzy number D_k derives from the center-average defuzzification method as follows: First, calculate the centroid of all points in P - call it P_c. Next, take the centroid of all the points in P whose abscissa is less than that of P_c i.e., $L = \{P_n | x(P_n) \leq x(P_c)\}$. Similarly, $G = \{P_n | x(P_n) \geq x(P_c)\}$ is the set of points whose abscissa is greater than that of P. The abscissae of three points P, L and G represent the resulting fuzzy distance D_k, formally described as:

$$D_k = (a, b, c) = \left(\left(\frac{\sum L_n}{|L|} \right)_x, (P_c)_x, \left(\frac{\sum G_n}{|G|} \right)_x \right) \tag{3}$$

In order to solve the non-linear system of Equations 2, in two fuzzy variables, the fuzzy variant of the iterative classical Newton method based on the Jacobian matrix [17,18] is used. To accomplish this, the fuzzy numbers are expressed in their parametric form $X = (\underline{X}, \overline{X})$ where $\underline{X}$ and $\overline{X}$ are continuous bounded non-decreasing and non-increasing, respectively, functions. These functions effectively represent the “left half” and “right half” of the membership function.

For a triangular membership function, such as defined in Equation 1 and depicted in Figure 2(a), a parametric representation in $r \in [0,1]$ is $X = (a + (b-a)r, c - (c-b)r)$. The system of Equations 2 are, therefore, represented in the parametric form. Without any loss of generality, assume that X and Y are positive. Then, each F_n in Equation 2 can be split into:

$$\begin{aligned} \underline{F_n} &= (\underline{X} - x_n)^2 + (\underline{Y} - y_n)^2 - \underline{D_n}^2 = 0 \\ \overline{F_n} &= (\overline{X} - x_n)^2 + (\overline{Y} - y_n)^2 - \overline{D_n}^2 = 0 \end{aligned} \tag{4}$$

The Jacobian J is constructed as:

$$J = \begin{bmatrix} \underline{F_1}_{\underline{X}} & \underline{F_1}_{\overline{X}} & \underline{F_1}_{\underline{Y}} & \underline{F_1}_{\overline{Y}} \\ \overline{F_1}_{\underline{X}} & \overline{F_1}_{\overline{X}} & \overline{F_1}_{\underline{Y}} & \overline{F_1}_{\overline{Y}} \\ \underline{F_2}_{\underline{X}} & \underline{F_2}_{\overline{X}} & \underline{F_2}_{\underline{Y}} & \underline{F_2}_{\overline{Y}} \\ \overline{F_2}_{\underline{X}} & \overline{F_2}_{\overline{X}} & \overline{F_2}_{\underline{Y}} & \overline{F_2}_{\overline{Y}} \\ \dots & \dots & \dots & \dots \\ \underline{F_n}_{\underline{X}} & \underline{F_n}_{\overline{X}} & \underline{F_n}_{\underline{Y}} & \underline{F_n}_{\overline{Y}} \\ \overline{F_n}_{\underline{X}} & \overline{F_n}_{\overline{X}} & \overline{F_n}_{\underline{Y}} & \overline{F_n}_{\overline{Y}} \end{bmatrix} = \begin{bmatrix} 2(\underline{X} - x_1) & 0 & 2(\underline{Y} - y_1) & 0 \\ 0 & 2(\overline{X} - x_1) & 0 & 2(\overline{Y} - y_1) \\ 2(\underline{X} - x_2) & 0 & 2(\underline{Y} - y_2) & 0 \\ 0 & 2(\overline{X} - x_2) & 0 & 2(\overline{Y} - y_2) \\ \dots & \dots & \dots & \dots \\ 2(\underline{X} - x_n) & 0 & 2(\underline{Y} - y_n) & 0 \\ 0 & 2(\overline{X} - x_n) & 0 & 2(\overline{Y} - y_n) \end{bmatrix} \tag{5}$$

The initial guess (X_0, Y_0) is computed from the average of the coordinates of the anchors. Define a matrix $F = [\underline{F_1}\ \overline{F_1}\ \underline{F_2}\ \overline{F_2}\ \dots\ \underline{F_n}\ \overline{F_n}]^T$. (X_0, Y_0) is updated in increments at every iteration by calculating an update matrix Δ using the matrices J and F evaluated at the current values of X and Y:

$$\Delta = \begin{bmatrix} \underline{h}(r) \\ \overline{h}(r) \\ \underline{k}(r) \\ \overline{k}(r) \end{bmatrix} = \begin{bmatrix} \underline{X}(r)^t & -\underline{X}(r)^{t-1} \\ \overline{X}(r)^t & -\overline{X}(r)^{t-1} \\ \underline{Y}(r)^t & -\underline{Y}(r)^{t-1} \\ \overline{Y}(r)^t & -\overline{Y}(r)^{t-1} \end{bmatrix} = -J^{-1}F \tag{6}$$

The process is repeated until Δ converges to 0 within ϵ.

3.2 Fuzzy Grid Prediction System (FGPS)

In mobile sensor networks with low anchor densities, it might frequently be the case that a node does not have enough anchors for multilateration. To address this problem we extend our fuzzy logic-based localization framework to predict an area, e.g., a cell in a grid, where the node might be. The idea is inspired from cellular systems [19]. We propose to virtualize the anchors, so that a node is within a set of Virtual Anchors at any point in time.

Consider the area in which the network is deployed to be subdivided into a grid of G cells, as depicted in Figure 3.2. We denote the probability that a node S is in a cell j ($j = 1 \dots G$) by p_j. To infer these probabilities, we construct a fuzzy system, whose input is the distance d_j between S and the center of cell j, and the output is a scalar $0 < p_j < 1$ for each j. The key idea is that the nearer a node is to the center of the cell, the more likely it is that the node can be found in that cell. A rule in our fuzzy system is as follows:

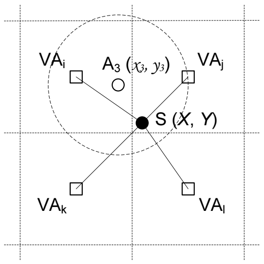


Fig. 4. A sensor S and the grid cells in its vicinity, is within radio range of anchor A_3

Rule i: **IF** (DIST$_{grd1}$ is D_{i1}) and ... and (DIST$_{grdG}$ is D_{iG}) **THEN** ($PROB_{grd1}$ is P_{i1}) and ... and ($PROB_{grdG}$ is P_{iG})

where D_{ij} is the fuzzy bin representing the distance between the node and the center of cell j, and P_{ij} is the fuzzy bin representing the probability that node S is in cell j.

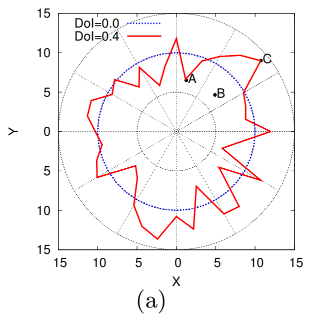
For each rule i, we calculate p_j by first fuzzifying d_j, applying it to the rule, and then defuzzifying the aggregate, as we described in Section 3.1. Once the most probable cell is found, the location of the node can be computed as the the intersection between this cell and the circle defined by the anchor.

It is paramount to remark that we can obtain p_j only if the node S has at least one anchor in its vicinity, i.e., we can estimate D_{ij}. The technique we propose for estimating D_{ij} is described in the Virtual Anchors section below. Before proceeding, we describe how to update p_j when no anchor is in the vicinity of node S. Since there is a high correlation between the current and previous cell a node is in, we construct a Recursive Least Squares [20] filter which predicts the cell in which the node S might be. For each cell j, we store m previous samples of p_j which serves as input to the filter.

Virtual Anchors. The fuzzy system requires that we calculate the distance from the node to the virtual anchor. We have to find the average distance instead, because we do not know the node's location. These average distances can be calculated only when at least one anchor is in the node's vicinity. The locus of the node around the anchor is a circle with radius as the radio range and center as the location of the anchor; the average distance to a virtual anchor can be easily calculated as the average of the distances between the virtual anchor and each point on this circle.

4 Performance Evaluation

Implementation. We implemented the FIS subsystem on EPIC motes running TinyOS 2.1 with an onboard CC2420 radio. The localization system was configured with 8 bins each for the distance and the RSSI, with ten preset rules. Upon receiving a message, the motes would read the RSSI from the radio and proceed to defuzzify it into a distance. This basic functionality was accomplished with



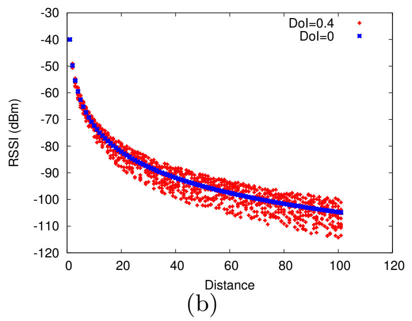


Fig. 5. (a) The DoI model with three points of interest: although A and B are equally distant, their RSS values differ significantly in our EDoI model; and (b) RSSI vs. distance for the radio model used in the simulator, at DoI=0.4 and 0

285 lines of code, occupying 19,932B in ROM and 1,859B in RAM (including the code required to send and receive messages over the radio). The FIS subsystem results were similar within rounding errors to those of the simulator's, with identical rules and binning. To the best of our knowledge, no mobile wireless sensor testbeds are available for public use. For the sake of repeatability of results and because of lack of mobile testbed infrastructure, we evaluate the performance of our localization scheme through extensive simulations. We use the data gathered from our static 42 node indoor testbed, as shown in Figure 8(b), for validating the FIS subsystem performance.

Simulation. Our proposed fuzzy logic-based localization system was implemented as an extension of the simulator provided by the authors of [8]. For performance evaluation, we compare FuzLoc, our solution, with MCL, MSL, Centroid and a "Perfect FuzLoc" algorithm, which is fuzzy multilateration with a theoretical ∼0% uncertainty in the ranges. These choices are justified by the fact that the we wanted to evaluate our solution against state of art Monte Carlo based solutions, as well as simpler techniques (e.g., Centroid), and demonstrate the advantages of our solution. We chose not to compare our solution against solutions that use additional hardware, such as OTMCL [11] which uses a compass.

Since our fuzzy logic-based localization technique makes use of the RSSI, we extended the DoI model [21]. In our EDoI model, the RSSI at the connectivity range predicted by the DoI model is associated with the minimum achievable received power at a receiver, i.e., receiver sensitivity. As shown in Figure 5(a), for points A and C (evaluated by the DoI model at the maximum radio ranges in two different directions), in our DoI model the RSSI is equal to the receiver sensitivity, -94dBm. For point B, we apply a log-normal fading model, such that the RSSI at point C (in the same direction as point B) is equal to receive sensitivity -94dBm. The predicted RSSI at point B is thus -60dBm. Formally, our EDoI model computes the RSSI, as follows:

$$RSSI(d) = S_i \frac{\log_{10} d}{\log_{10}[r(1 + DoI \times rand())]} \tag{7}$$

where S_i is the receiver sensitivity, r is the ideal radio range, DoI is the radio degree of irregularity and $rand$ is a random number $\mathcal{U}[0,1]$.

We simulate a set (N) of 320 sensor nodes deployed in a 500 × 500 area. Of the 320 nodes deployed, 32 nodes are designated anchors (set S). The radio range (r) of a node is 50 and the default DoI is 0.4. We chose these simulation parameters for consistency with results reported in [8], [9]. The default receiver sensitivity (S_i) is -94dBm, and a plot depicting the predicted RSSI by our EDoI model, is shown in Figure 5(b). The default maximum node velocity is to 0.2r. This velocity has been reported in [8] and confirmed in [11], to be optimal. We investigate the performance of all solutions for node velocities up to 0.5r. The results are averaged over all nodes over 10 runs, with each node taking 50 steps per run for a total of atleast 16000 trials per data point. The default setup uses 10 fuzzy triangular bins and the defuzzification method is center-average. The fuzzy location is defuzzified into a crisp location by considering only the center values of the abscissa and the ordinate.

Radio Irregularity. We performed simulations for different DoI values with all other parameters kept constant. Figure 6(a) depicts our results, indicating the deterioration in localization accuracy of MCL and MSL. The effect of compounded errors due to polluted samples (incorrectly computed locations which are used to compute the location in future iterations) has been investigated as the "kidnapped robot problem" [22] in robot localization. The kidnapped robot test verifies whether the localization algorithm is able to recover from localization failures, as signified by the sudden change in location due to "kidnapping". It has been shown that such uncorrected algorithms collapse when the observed sample is far from the estimated sample. MSL demonstrates an even more pronounced effect, since it also uses non-anchor neighbors for filtering, thus leading to even more pollution.

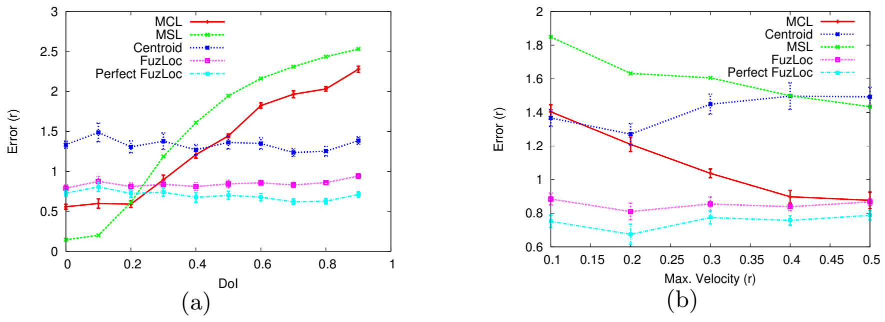
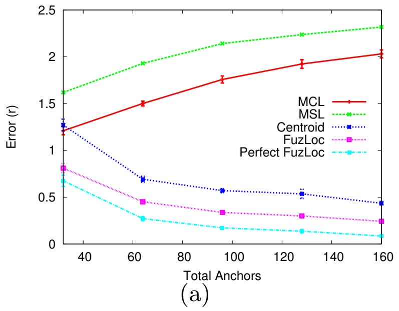


Fig. 6. Localization accuracy as affected by (a) DoI (N=320, S=32, v=0.2r); and (b) maximum node velocity (N=320, S=32, DoI=0.4)

Maximum Node Velocity. We investigate the effect of maximum node velocity on localization accuracy, for velocities up to 0.5r, a reasonably fast moving speed. The performance results are depicted in Figure 6(b). MCL and MSL assume that nodes know their maximum velocity. Hence, they use the velocity as a filtering condition, which improves their performance. Moreover, high velocity means having more anchors to filter against, leading to the freshening of samples at every instance. Figure 6(b) shows that MCL and MSL decrease their localization error from 1.4r to 0.9r, and 1.9r to 1.4r, respectively. Since Centroid and FuzLoc do not use the velocity, their performance is not expected to improve. Figure 6(b) indicates that their performance is not deteriorating.



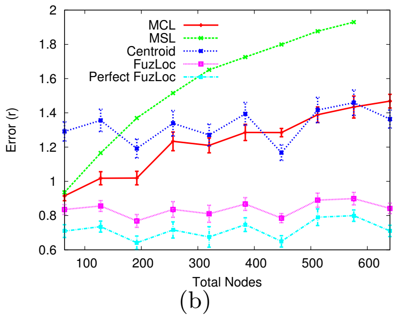


Fig. 7. Localization accuracy as affected by (a) anchor density at DoI=0.4 (N=320, v=0.2r); and (b) node density (S=32, v=0.2r, DoI=0.4)

Anchor Density. Anchor density is a critical parameter for anchor-based localization schemes. Figure 7(a) displays the impact of anchor density on the localization schemes where the number of anchors varies from 10% (32 anchors) to 50% (160 anchors), and the DoI is constant at 0.4. The accuracy of MCL and MSL deteriorates because an increase in anchor density is associated with an increase in the number of polluting sources. The mismatch of observed and actual radio ranges causes spurious anchors to appear as node's direct and indirect seeds. MSL considers non-anchor neighbors, hence it experiences higher pollution. Centroid performs better with increasing anchor density, as expected. FuzLoc also has a decrease in localization error, with a larger number of anchors. We observe that FuzLoc is not greatly affected by DoI and ranging errors.

Node Density. For this performance evaluation scenario we maintained the percentage of anchors fixed at 10%. As shown in Figure 7(b), the evaluated algorithms either suffer or are unaffected. None of the localization algorithms benefits from an increase in the node density. As shown, Centroid and FuzLoc are not substantially affected, except by the inherent randomness in simulation. MCL considers indirect seeds for sampling, hence a high node density means more anchors are misreported as indirect seeds. MSL considers non-anchor neighbors, hence at high node densities, it experiences a huge amount of sample pollution. While non-anchor neighbors help MSL to improve accuracy at low DoI, they become harmful at higher DoI values.

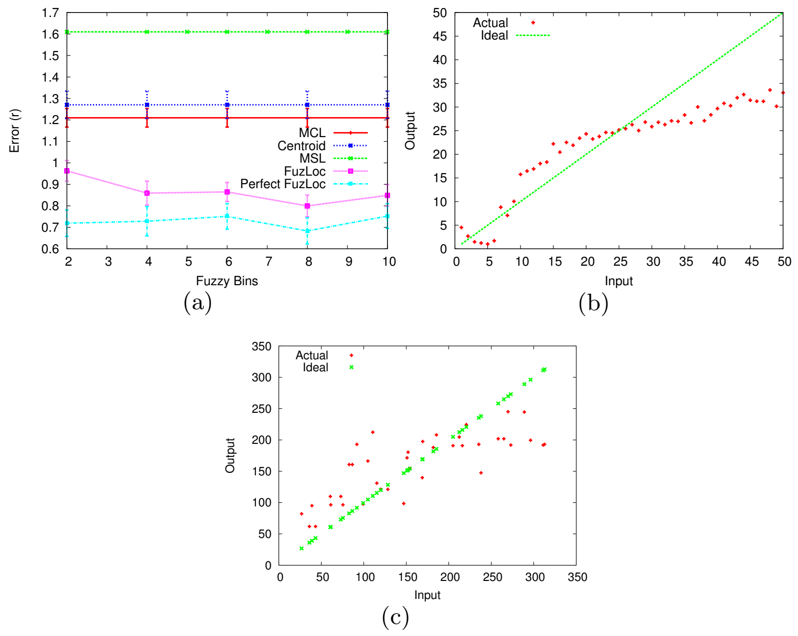


Fig. 8. (a) Localization accuracy as affected by the number of fuzzy bins (N=320, S=32, v=0.2r, DoI=0.4); (b) Performance of the FNLS FIS subsystem based on the simulated EDoI radio model; and (c) Performance based on real data gathered from our indoor testbed

Number of Bins. The number of bins in the fuzzy system is a design parameter - the greater the number of bins, the higher the accuracy of the system. Our evaluation of the influence of the number of bins is depicted in Figure 8(a). As shown, as the number of bins increases, the localization error of FuzLoc decreases. This is because more and more RSSs find a bin with high membership. The change in the number of bins, is expected to not affect MCL, MSL, Centroid, or even Perfect FuzLoc. Figure 8(a) shows that the aforementioned schemes remain invariant whereas FuzLoc experiences decreasing error with an increase in the number of bins.

FIS Performance. Real data gathered from our testbed (42 nodes deployed over a 600 sq.ft. indoor multipath environment) was used to evaluate the radio model and the FIS used in the simulator, by comparing the accuracy of the FIS subsystem as shown in Fig. 8(b) and Fig. 8(c). Figure 8(b) shows the performance of the FNLS FIS engine. Input distance on the X axis is translated into an RSS which is then defuzzified into a distance on the Y axis. Figure 8(c) is similar except that the RSS is extracted from the real dataset. After training the system with 30 random RSS-Distance pairs, RSS values deduced from distances were fed into the system so that a distance should be inferred. The straight line shows the ideal case. The FIS is somewhat efficient, except at the fringes, where the fuzzy system was not found to be trained due to the limited number of rules. With more rules, the system becomes more and more efficient. This result indicates the resiliency of FuzLoc to ranging errors.

Overhead. A typical FIS does not require significant storage capacity. If there were 8 bins, for example, a single byte could represent a bin. Hence, each FNLS rule requires just 2 bytes of storage. Typically, an anchor creates approximately 30 rules during the period of deployment which translates to 60 bytes of storage. The FGPS FIS however, requires 50 bytes for each rule (25 bins in the input, 25 in the output). Note that regular nodes do not store rules, only the anchors store rules. Moreover, due to the nature of the triangular bin shapes, simple calculations are required in order to (de)fuzzify. MCL requires at least 50 weighted samples for low localization error. Centroid does not store any history and thus has the smallest storage requirement. Amorphous stores announcements made by the anchors which are flooded throughout the network. If there are 320 nodes, 32 of which are anchors, MCL requires each node to store 50 samples - $(50 \times 4 \times 2 \times 320) = 128,000$B. Fuzzy on the other hand requires around 1,500 bytes for FGPS and around 60 for FNLS, which sums up to roughly 50% of the storage MCL requires. We note here that communication overhead is very similar among all evaluated localization techniques.

5 Conclusions

We have proposed FuzLoc, a fuzzy logic based localization method for harsh environments. The constituent systems use fuzzy multilateration and a grid predictor to compute the location as an area. Our method has been evaluated based on a variety of metrics. They prove that it is resistant to high DoI environments while providing a low localization error without any extra hardware. Only anchors need to have a slightly higher storage requirement. A deployment with more anchors at high DoI decreases the error. FuzLoc's principle limitation is that it does not work very well in static networks unless the anchors have pre-loaded fuzzy rules or the anchors have a large radio range. The distributed protocol requires some amount of data to be transmitted. This could be problematic in networks populated by resource-constrained nodes. Future work includes a module for iteratively redesigning the fuzzy bins to optimize based on network characteristics.

Acknowledgement. This work was funded in part by NSF grant CNS 0923203. The authors also wish to thank Mike George for his help.

References

1. Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., Welsh, M.: Deploying a wireless sensor network on an active volcano. IEEE Internet Computing 10(2) (March-April 2006)
2. George, S., Zhou, W., Chenji, H., Won, M., Lee, Y., Pazarloglou, A., Stoleru, R., Barooah, P.: A wireless ad hoc and sensor network architecture for situation management in disaster response. IEEE Communications Magazine (March 2010)

3. He, T., Krishnamurthy, S., Luo, L., Yan, T., Gu, L., Stoleru, R., Zhou, G., Cao, Q., Vicaire, P., Stankovic, J., Abdelzaher, T., Hui, J., Krogh, B.: VigilNet: an integrated sensor network system for energy-efficient surveillance. ACM Trans. Sens. Netw. 2(1) (2006)
4. Girod, L., Lukac, M., Trifa, V., Estrin, D.: A self-calibrating distributed acoustic sensing platform. In: SenSys (2006)
5. Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: MobiCom (2000)
6. Kusy, B., Sallai, J., Balogh, G., Ledeczi, A., Protopopescu, V., Tolliver, J., DeNap, F., Parang, M.: Radio interferometric tracking of mobile wireless nodes. In: MobiSys (2007)
7. Whitehouse, K., Karlof, C., Culler, D.E.: A practical evaluation of radio signal strength for ranging-based localization. Mobile Computing and Communications Review 11(1) (2007)
8. Hu, L., Evans, D.: Localization for mobile sensor networks. In: MobiCom (2004)
9. Rudafshani, M., Datta, S.: Localization in wireless sensor networks. In: IPSN (2007)
10. Baggio, A., Langendoen, K.: Monte carlo localization for mobile wireless sensor networks. Ad Hoc Netw. 6(5) (2008)
11. Martins, M., Chen, H., Sezaki, K.: OTMCL: Orientation tracking-based monte carlo localization for mobile sensor networks. In: INSS (June 2009)
12. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less low cost outdoor localization for very small devices. IEEE Personal Communications Magazine (2000)
13. Zhong, Z., He, T.: Achieving range-free localization beyond connectivity. In: SenSys (2009)
14. Shen, X., Mark, J.W., Ye, J.: Mobile location estimation in CDMA cellular networks by using fuzzy logic. Wirel. Pers. Commun. 22(1) (2002)
15. LeBlanc, K., Saffiotti, A.: Multirobot object localization: A fuzzy fusion approach. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics 39(5), 1259–1276 (2009)
16. Saffiotti, A.: The uses of fuzzy logic in autonomous robot navigation. Soft Computing-A Fusion of Foundations, Methodologies and Applications 1(4), 180–197 (1997)
17. Kelley, C.T.: Fundamentals of Algorithms - Solving Nonlinear Equations with Newton's Method. SIAM, Philadelphia (2003)
18. Shokri, J.: On systems of fuzzy nonlinear equations. Appl. Math. Sci, Ruse (2008)
19. Shen, X., Mark, J.W., Ye, J.: User mobility profile prediction: an adaptive fuzzy inference approach. Wirel. Netw. 6(5) (2000)
20. Hayes, M.: Statistical digital signal processing and modeling. Wiley India Pvt. Ltd, Chichester (2008)
21. He, T., Huang, C., Blum, B.M., Stankovic, J.A., Abdelzaher, T.: Range-free localization schemes for large scale sensor networks. In: MobiCom (2003)
22. Engelson, S., McDermott, D.: Error correction in mobile robotmap learning. In: ICRA (1992)

AEGIS: A Lightweight Firewall for Wireless Sensor Networks

Mohammad Sajjad Hossain and Vijay Raghunathan

School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47906
{sajjad,vr}@purdue.edu

Abstract. Firewalls are an essential component in today's networked computing systems (desktops, laptops, and servers) and provide effective protection against a variety of over-the-network security attacks. With the development of technologies such as IPv6 and 6LoWPAN that pave the way for Internet-connected embedded systems and sensor networks, these devices will soon be subject to (and need to be defended against) similar security threats. As a first step, this paper presents AEGIS, a lightweight, rule-based firewall for networked embedded systems such as wireless sensor networks. AEGIS is based on a semantically rich, yet simple, rule definition language. In addition, AEGIS is highly efficient during operation, runs in a transparent manner from running applications, and is easy to maintain. Experimental results obtained using real sensor nodes and cycle-accurate simulations demonstrate that AEGIS successfully performs gatekeeping of a sensor node's communication traffic in a flexible manner with minimal overheads.

Keywords: Wireless Sensor Networks, Firewall, Network Overlay.

1 Introduction

The rapid proliferation of networked embedded systems such as wireless sensor networks (WSNs) is expected to further accelerate in the near future due to the development of enabling technologies such as 6LoWPAN [11,12] and lightweight TCP/IP stacks for embedded microcontrollers [5], which aid in realizing the vision of an *Internet-of-Things* [8]. However, this widespread adoption will bring with it additional challenges. As we learnt (and continue to learn) in the case of personal computers (PCs), a networked computing system can be a nightmare to secure because it is exposed to a wide variety of over-the-network security attacks. It is only natural that, going forward, networked embedded systems and WSNs (especially if connected to the Internet) will inherit the same security problems that plague their networked counterparts in the PC world.

Firewalls [4] form an integral component of the security architecture of today's networked PC systems. They provide an effective defense mechanism against many over-the-network attacks by performing careful gatekeeping over the communication traffic entering and exiting a system. A firewall provides a clean interface to restrict and control bidirectional access to/from an individual application, an entire device, or even a portion of a network. In addition, a firewall can also be judiciously used to enforce various communication resource management policies, as we will demonstrate.

Even though firewalls have been a must-have feature of networked PCs for a long time and have been extensively explored in that context, WSN systems still lack even the most basic form of a firewall. Although the resource constrained nature of these systems imposes limiations on what security mechanisms can be deployed, we believe that, going forward, traffic gatekeeping will be an absolute necessity for WSNs. To address this need, this paper presents Aegis[1], a software firewall architecture for networked embedded systems such as WSNs. Aegis provides stateless, rule-based gatekeeping over the network interface of a sensor node, akin to a typical packet-filtering firewall in conventional PC systems. The following are the key novel attributes of Aegis:

- Aegis features a semantically rich, yet easy to use, rule definition language that allows users to easily construct complex firewall policies using a sequence of simple packet filtering rules.
- Typically, packet-filtering firewalls operate on a given rule file as input. However, such an architecture is inefficient, and often even infeasible, for sensor nodes because the rule file has to be stored either in RAM, which is a scarce resource in WSNs, or in external data flash, which has a high energy cost of read/write. Aegis addresses this issue by transforming the rule file into executable binary code. The binary code is stored and executed from the microcontroller's on-chip program flash, which is typically more abundant than RAM and lower in terms of energy per access than external data flash. In addition, such a transformation improves computational efficiency by making it easy to identify which rules are relevant to a data packet being filtered and ensuring that irrelevant rules are not even looked at. Finally, various optimizations are applied during the transformation and code generation process, which eliminate the need for firewall rule-set optimizations used otherwise (*e.g.,* eliminating/resolving rule redundancies and conflicts) [18,9].
- Aegis is completely transparent to other software modules[2] whose communication is being firewalled, and requires no access to the source code or binary of those modules. Further, when used in a modular operating system (in our prototype implementation, SOS [10]), updating firewall policies in Aegis becomes similar to updating any other binary module, making firewall maintenance much simpler.
- In addition to providing basic security functionality, Aegis also enables a variety of other interesting applications (*e.g.,* creating logical overlay networks), as shown in Section 4.3.

[1] In Greek mythology, Aegis is a protective shield worn by the god Zeus and the goddess Athena.

[2] In a modular operating system (*e.g.,* SOS [10]), software modules can be thought of as individual applications running on top of the operating system.

We have designed and implemented a fully functional prototype of Aegis using the SOS operating system and evaluated it using experiments conducted on real wireless sensor nodes as well as through cycle-accurate simulations.

2 Firewall Design Principles - From a WSN Perspective

One of the main tenets of network security is security *in-depth* or *multi-layered* security. As an embodiment of this principle, in a network of PCs, running a gateway-level firewall (*e.g.,* on a home router or wireless access point) does not eliminate the need for a software firewall on the PCs connected to the gateway. Similarly, while firewalls running on the gateways to sensor networks will serve as a first line of defense, they will not be sufficient by themselves. For example, an adversary who is within radio range of a sensor node can directly send packets to it without having to go through the gateway, thus bypassing the gateway-level firewall altogether. Aegis enables us to apply this principle in WSNs by enabling careful traffic gatekeeping on individual sensor nodes.

Existing design principles for firewalls in traditional networks are not applicable to wireless sensor nodes due to their resource limited nature. Hence, any firewall design targeting wireless sensor nodes should take into account these limitations while at the same time providing a true firewall experience. It has to be *semantically rich* in order to provide the same level of flexibility as desktop PC firewalls. Such flexibility can be achieved using context free languages to specify firewall policies. To be a viable option, a firewall for WSNs must be *lightweight and efficient* in terms of RAM and ROM usage as well as computational demand. *Transparency* is another objective that needs to be met by a good firewall design. It is important that the binary or source code of a module or application stay unmodified due to the presence or absence of a firewall. Finally, the firewall should be *easy to maintain.* For example, if the firewall rules or policies change, the operation of a node or part of the network should not be interrupted to accommodate the changes. Also, since sensor nodes are often deployed in remote locations, applying such updates should ideally not require physical access to the devices.

Apart from these, the requirements of the specific application domain may pose several other challenging design goals. Examples include secure dissemination of the firewall itself, tamper resistance of the firewall, *etc.*

3 The Aegis Architecture and Implementation

The detailed description of the architecture and implementation of Aegis is presented in this section.

3.1 Overview

Aegis offers stateless, application layer firewall-like protection for sensor networks and embedded systems in general. It was designed mostly for operating

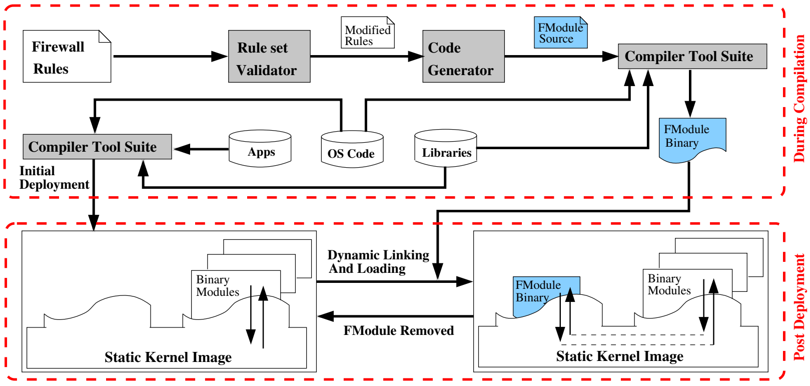


Fig. 1. Overview of the AEGIS architecture

systems that provide module-level granularity in their run time execution systems. In such OSs, each component can be thought of as a separate module in the operating environment. Operating systems such as SOS [10], Contiki [6] and MANTIS [2] provide such an architecture. For a firewall to function properly, it has to have access to all incoming and outgoing packets and messages. One way of achieving this would be to run each module within a separate sandbox. In our previous work, we developed an architecture, called HERMES [14], which provides a flexible environment for such sandboxing techniques. HERMES provides fine grained control over a module's behavior by intercepting all system calls made by it. This flexibility can be exploited to secure a module by controlling its incoming and outgoing packets and messages. Unlike HERMES, AEGIS is also targeted to provide node level (*i.e.*, interaction between two nodes) security rather than module level (*i.e.*, interaction of a module with another module within the same node or another node in the network) security only. It is true that by controlling all the existing modules in the system, we can achieve a node level control mechanism. But such a solution will incur unnecessary overhead since sandboxing a module requires memory and CPU cycles.

Instead, we take another approach in AEGIS. Here, we intercept all incoming and outgoing packets and messages in the kernel and forward the meta-data (*e.g.*, source, destination, *etc.*) to another module (we refer to it as *FModule*), which analyzes the meta-data. Based on the result of the analysis by FModule, the kernel either allows or denies the communication in question. Such an approach meets the objective of achieving typical firewall functionality. In addition, AEGIS also provides control over module-module communication. This allows us to apply various resource management policies for individual modules, as we demonstrate later.

In the following subsections, we describe the key components and implementation details of AEGIS. Figure 1 shows the interaction among the key AEGIS components.

3.2 SOS Operating System

SOS [10] is a modular operating system for sensor nodes that consists of a set of dynamically loadable modules on top of a common kernel. SOS provides dynamic memory handling, module loading and unloading during run time, garbage collection and a cleaner interface for asynchronous module communication through message passing. Figure 2 shows how SOS modules communicate with other modules in the same node or another node in the network through message passing. We use the terms "message" and "packet" interchangeably from now on. SOS also provides mechanisms for synchronous communication among modules through function subscription [10]. Like most other operating systems being used in sensor networks, SOS, in its original form, does not provide memory protection. Even though we implemented AEGIS in SOS, it should be noted that it is still portable to other sensor network operating systems (Section 3.8).

3.3 Intercepting Communication

FModule, an SOS module stitched from the input rule set, provides two methods: `is_outgoing_allowed` and `is_incoming_allowed` which take communication meta-data as input for outgoing and incoming messages respectively. They return `true` if it is to be allowed and `false` otherwise. The communication meta-data contains source and destination module identifiers as well as source and destination node addresses.

In SOS, network communication is performed using one of the `post_*` kernel API calls. AEGIS uses a modified SOS kernel, where, at the beginning of these function calls, `is_outgoing_allowed` provided by FModule is invoked. It should be noted that in SOS, the kernel is not a module and hence can not subscribe to functions provided by FModule or any other module. So, the modified kernel accesses the appropriate function pointer from a *system jump table* [10] in order to invoke the method `is_outgoing_allowed`.

Similarly, incoming messages are inspected by calling `is_incoming_allowed` from the appropriate place in the network stack (we intercepted `handle_incoming_msg()`).

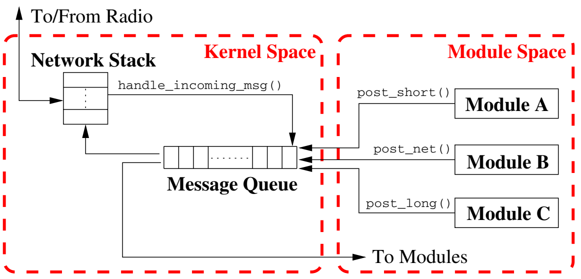


Fig. 2. Communication in SOS using message passing

3.4 Rule Specification

Like firewalls used in traditional networking environments, we take a declarative approach in AEGIS for defining firewall rules. These rules are defined by the grammar, G (Figure 3). FModule is automatically generated from the rule specification (Section 3.6). Note that the default behavior in AEGIS is to allow all communication.

```
<rules>      ::= "BEGIN" <rule-list> "END”
<rule-list>::= <rule> | <rule> < rule-list>
<rule>       ::= <address> <comm-dir> <address> <access>
<address>    ::= <id-list> | "*" | ".”
<comm-dir>   ::= "IN" | "OUT”
<access>     ::= "ALLOW" | "DENY”
<id-list>    ::= <id> | <id> <id-list>
<id>         ::= "N" <int-lit> | "M" "(" <m-args> ")" <int-lit>
<m-args>     ::= <int-list> | "*" | ".”
<int-list>   ::= <int-lit> | <int-lit> "," <int-list>
<int-lit>    ::= <digit>+
<digit>      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Fig. 3. Grammar used to define AEGIS rules. Here, `.` (dot) means the current node and `*` (asterisk) is used to refer to all the nodes in the network

3.5 Rule Set Validation

A firewall policy may often contain conflicting or prohibited rules due to human errors. Since AEGIS incorporates rules into a binary, it is difficult to update that binary in the event of an error as it requires an expensive code update procedure. Thus, it is important that the rule set used to generate FModule is free from any conflicting or prohibited rules. For example, consider these two rules: `* OUT N 6 ALLOW` and `* OUT N 6 DENY`. Here, `N` refers to the nodes and `M` refers to the modules in a node. Other details about the syntax can be found in Figure 3. The first rule allows all traffic towards node 6 while the latter does not. To avoid this conflict, at least one of them needs to be eliminated. Examples of prohibited rules can be those which may disrupt communication to or from modules (referred as *prohibited modules*) or nodes that are an integral part of the operation of the network (*e.g.*, the SOS module that is responsible for dynamic loading).

An *atomic rule* is a rule which addresses packets from and to a particular module in a particular node, *e.g.*, `M(2) 3 OUT M(3) 2 ALLOW`. On the other hand, `M(2, 3) 3 OUT N 2 ALLOW` is not an atomic rule. Each non-atomic rule can be broken into two or more atomic rules. We refer to this as *rule decomposition.* Table 1 shows a few examples of rule decomposition including the aforementioned non-atomic rule. The decomposition process allows us to resolve conflicts at such a granularity that we do not need to discard a rule in its entirety unless needed. The decomposed rule set can be much bigger than the original rule set. We can keep the cardinality of this set small by carefully assigning IDs to modules and nodes so that the values of `MAX_MODULE_ID` and `MAX_NODE_ID` (Table 1) remain

Table 1. Decomposing firewall rules into atomic rules in AEGIS

Original Rule	Decomposition
`N 3 OUT M(6) 1 ALLOW`	`M(3) 1 OUT M(6) 1 ALLOW, M(3) 2 OUT M(6) 1 ALLOW,..., M(3) MAX_MODULE_ID OUT M(6) 1 ALLOW`
`M(2, 3) 3 OUT N 2 ALLOW`	`M(2) 3 OUT M(2) 1 ALLOW, M(2) 3 OUT M(2) 2 ALLOW,..., M(2) 3 OUT M(2) MAX_MODULE_ID ALLOW,` `M(3) 3 OUT M(2) 1 ALLOW, M(3) 3 OUT M(2) 2 ALLOW,..., M(3) 3 OUT M(2) MAX_MODULE_ID ALLOW`
`* OUT N 4 DENY`	`N 1 OUT N 4 DENY, N 2 OUT N 4 DENY,..., N MAX_NODE_ID OUT N 4 DENY` These rules are further decomposed like the previous examples.

as small as possible. Moreover, since we perform this validation process in a desktop environment, we have access to enough resources to handle this.

In order to define the rule validation process more formally, we use the following notation:

P : The initial rule set	X : Set of the prohibited modules
R_i : The ith rule in P	r^i_j : The jth atomic rule after decomposing R_i
s^i_j : Source module in r^i_j	t^i_j : Destination module in r^i_j
d_i : Direction in R_i (`IN`/`OUT`)	a_i : Access specification in R_i (`ALLOW`/`DENY`)

The following cases are considered while validating P:

1. If $\exists i, j \ (s^i_j \in X \vee t^i_j \in X)$ then discard r^i_j.
2. $\exists i, j, k^i, k^j \ (i < j \wedge s^i_{k^i} = s^j_{k^j} \wedge t^i_{k^i} = t^j_{k^j} \wedge d_i = d_j \wedge a_i \neq a_j)$ and both $r^i_{k^i}$ and $r^j_{k^j}$ are still not discarded, discard $r^j_{k^j}$. It should be noted that there can not be any conflicting atomic rules within a single rule (since $a_i \neq a_i$ is always false), hence the case where $i = j$ is not required.

Note that these cases do not handle redundant rules but only conflicting and prohibited rules. This omission is deliberate because we show in the next section that the code generation mechanism in AEGIS is immune to possible redundancies in P. There is a caveat with this validation process: it can only detect and resolve *static conflicts* among the rules. However, there can be conflicts arising during the run-time as well. Rules that address the current node (using `.` (dot) as the source/destination) can create such conflicts since the node ID is not available during the compile time. We generate FModule code in such a way that the rule that is defined earlier in the input file stays effective in such cases.

3.6 Code Generation and Optimization

A straightforward approach in designing a firewall will be to use a rule file as the input to FModule. Under this model, whenever a transmission gets directed towards FModule, it reads the rule file to check whether the transmission should be allowed or denied. Besides simplicity, it has other benefits. For example, policy update (*e.g.*, adding a new rule) can be done in an incremental fashion resulting in smaller data transfer across the network. However, there are a few serious drawbacks as well. Most of the operating systems for sensor nodes do not provide any easy mechanism for securing a file or memory protection [15]. So, malicious modules may get access to the rule file and manipulate it according to their needs. Optimizing the rule set will be almost impossible in such an approach as it would be computationally very expensive for sensor nodes. Also, each time FModule receives the meta-data of a packet or message, it has to read the input file resulting in a large number of read operations in its life time. Such operations are prohibitively expensive for an environment where resources are extremely limited.

In contrast, AEGIS addresses these drawbacks by generating a binary, called FModule, from the input rule file and uploading the binary to the nodes. Disseminating this binary is likely to be more expensive than doing an incremental update of the rule set used in the previous approach. Since such policy changes are infrequent, this does not become a big issue. However, this allows us to perform complex code analysis and optimizations in powerful computing environments. It also eliminates the need for expensive read operations from the flash memory during run-time. Our code generation technique avoids the need for complex rule optimization techniques [18,9] otherwise found in traditional firewalls.

After removing the *static conflicts* and *prohibited rules* from the rule specification (Section 3.5), we generate C-code which will be compiled into FModule. A naive code generation technique can produce an inefficient executable. Hence, we apply simple optimizations as shown in Figure 4. Here, the most straightforward conversion of the rules results in multiple `if-then-else` code blocks. Sometimes it may be possible to fuse some of them into a fewer number of blocks, as shown for `is_outgoing_allowed` in Figure 4. It is also useful to guard against redundant and duplicate rules since it would void the effect of such redundancies. Another optimization applied is the use of `switch-case` statements. Since nodes in a sensor network typically run only a small number of modules, sometimes the use of `switch-case` based on *module id* can lead to efficient binaries. For small networks, use of `switch-case` based on *node id* can also be beneficial.

We claimed earlier that unlike traditional firewalls, we do not perform any optimization on the input rule set. This is because we generate FModule source in such a way that it eliminates the necessity of such optimizations any more. Figure 4 shows an example of this inside the method `is_incoming_allowed`. The generated code there removes the redundancy introduced by the fourth rule in the actual input rule set.

As a proof of concept, we apply the above optimizations to our generated code. However, in principle, other complex optimization techniques can be part of AEGIS. Both *rule set validation* and *code generation and optimization* are automated and implemented using standard Unix utilities `lex` and `yacc`.

3.7 Meeting the Design Principles

In Section 2, we outlined a few firewall design principles aimed at resource scarce embedded systems such as WSNs. We designed AEGIS to align with those principles as closely as possible. With the use of a semantically rich rule specification

```
...
M(.) 5 OUT N 5 N 6 DENY      M(.) 5 OUT N 4 N 7 DENY
M(*) 6 IN . ALLOW            M(3) 6 IN . ALLOW
```

(a) Firewall rules

```
...
int8_t is_outgoing_allowed(sos_pid_t did,
                           sos_pid_t sid,
                           uint16_t daddr,
                           uint16_t saddr) {
    /* Rule #1 */
    if (sid == 5 && saddr == ker_id()) {
        if (daddr == 5 || daddr == 6) {
            return DENY;
        }
    }
    /* Rule #2 */
    if (sid == 5 && saddr == ker_id()) {
        if (daddr == 4 || daddr == 7) {
            return DENY;
    }
    return ALLOW;
}

int8_t is_incoming_allowed(...) {
    /* Rule #3 */
    if (sid == 6) {
        if (daddr == ker_id()) {
        return ALLOW;
    }
    /* Rule #4 */
    if (sid == 6 && saddr == 3) {
        if (daddr == ker_id()) {
            return ALLOW;
        }
    }
    ...
    return ALLOW;
}
```

(b) Code generated without optimization

```
...
int8_t is_outgoing_allowed(sos_pid_t did,
                           sos_pid_t sid,
                           uint16_t daddr,
                           uint16_t saddr) {
    ...
    switch (sid) {
        case 5:
            if (saddr == ker_id() && daddr >= 4
                && daddr <= 7) {
                return DENY;
            }
            break;
        ...
    }
    return ALLOW;
}

int8_t is_incoming_allowed(...) {
    ...
    switch (sid) {
        case 6:
            if (daddr == ker_id()) {
                return ALLOW;
            }
            break;
        ...
    }
    ...
    return ALLOW;
}
```

(c) Code generated with optimization

Fig. 4. Code optimization in Aegis

language (Section 3.4), Aegis allows users to define a diverse set of policies. The code generation technique in Aegis makes sure that redundant rules in the input rule set do not increase the code size. The memory overhead incurred by Aegis is independent of the number of modules present in the system. For example, we micro-benchmark Aegis (Table 2) in Section 4 and find it to increase the ROM usage of SOS only by 2.7 KB. By incorporating rules into a binary (FModule), Aegis eliminates huge amount of expensive *read* operations otherwise needed in the case where rules are read from an input file each time a check is performed. Aegis provides transparency from user modules by not requiring any modification of them and by intercepting communication in the kernel. Maintenance of Aegis is easy when it is used in an operating system with support for dynamic loading and linking (*e.g.*, SOS [10]).

3.8 AEGIS for Other Operating Systems

Even though SOS was chosen as Aegis' development platform, other modular operating systems like Contiki [6] and MANTIS [2] can also be used to run Aegis effectively. Operating systems like TinyOS [1] that do not have a modular structure can still benefit from Aegis. Even though control over module level communication is no longer possible, Aegis can still provide necessary firewall functionality to control node level communication. For example, active messages (AM) [3] for MicaZ motes running TinyOS can be controlled from the AM implementation for the CC2420 radio (component `CC2420ActiveMessageP`). This way, user applications need not be modified to enforce firewall policies. Updating the firewall can be done in the form of standard code dissemination [17].

4 Experimental Results

To evaluate our prototype implementation of Aegis, we conducted a series of experiments using a cycle-accurate simulator [22] as well as a small testbed of MicaZ motes. Our first experiment evaluated the overheads of Aegis while the second and third experiments evaluated Aegis' ability to perform gatekeeping at the node-level and module-level, respectively.

4.1 Overhead Analysis

Our first experiment was a micro-benchmark evaluation of our Aegis implementation using the Avrora cycle-accurate simulator to measure its overheads. In all the runs of this experiment, there were three modules pre-installed in SOS, namely Surge, TreeRouting and PhotoTempSensor, all available with the standard SOS installation. We generated an FModule for this experiment from these two rules: `N 2 N 3 IN * DENY` and `* OUT N 2 N 3 DENY`.

Table 2 shows the increase in total RAM (Data + BSS + Heap + Stack) and ROM usage due to Aegis. As shown in the table, the ROM usage increased by 6.5% and the RAM usage increased by 7.6%. As described earlier, FModule provides two methods: `is_outgoing_allowed` and `is_incoming_allowed` to filter all outgoing and incoming communications respectively. We found the overhead of calling these two functions to be 15 and 24 clock cycles, respectively, for the rule set listed above. Note that this overhead will vary based on the complexity of the rule set used (*e.g.,* an empty `is_outgoing_allowed` method with only a return statement takes 9 cycles). Table 2 also lists the overhead of invoking SOS' messages sending functions. The overhead was more than that of just invoking the two FModule methods mentioned above. This is due to the look up time for FModule and the appropriate function pointer from the SOS system jump table inside the kernel. The overhead of having Aegis but no FModule was between

Table 2. Memory footprint of various combinations of SOS and Aegis along with overhead of calling message sending functions

OS Configuration	Memory (bytes)		SOS Function Call (# of cycles)		
	RAM	ROM	`post_short`	`post_long`	`post_link`
Plain SOS	3,604	45,114	262	392	585
SOS + Aegis + no FModule	3,755	47,856	425	571	742
SOS + Aegis + FModule	3,881	48,048	585	790	967

163 and 185 clock cycles. The addition of an FModule further increased the execution time by approximately the same amount.

4.2 Controlling Tree Routing

In our second experiment, also conducted using Avrora, we used AEGIS with SOS running Surge [24], a sensor data collection application. Surge uses a distributed tree routing protocol that builds a tree rooted at the base station and every Surge node forwards their collected data towards that base station. *TreeRouting* is an SOS module that implements this routing protocol. This experiment demonstrates a scenario where selected nodes have to be cut off from the network for some reason (*e.g.,* the nodes display suspicious behavior or are known to have a hardware or software bug). Figure 5 shows the experimental topology used. All nodes were forwarding their data to node 1, which was the base station of this topology. The Surge module on each node was generating a data packet in every 8 seconds. After TreeRouting stabilized (after approximately 100 seconds) and had a route established for each node towards the base station, we updated the firewall in all the nodes with the rules: `N 2 N 3 IN * DENY` and `* OUT N 2 N 3 DENY`. This blocked all nodes from sending/receiving packets to/from nodes 2 and 3. As a result, TreeRouting re-adjusted and some nodes started sending data following different routes (Figure 5). Later on, we lifted the restriction on node 2 by replacing the rules with these: `N 3 IN * DENY` and `* OUT N 3 DENY`. Since SOS does not provide the ability to install or update binaries on a selected node only, all nodes received the updated firewall rules. Soon after this, node 2 became part of the routing tree again while node 3 still remained isolated. This simulation was run for 45 minutes in Avrora. To show how the packet delivery latency at the base station was affected, we consider packets that originated from node 5. Latency was calculated as the time a packet took to traverse from the physical layer of the source to that of the base station. Without any firewall active, the average delivery latency was calculated to be 39.3 ms. When both node 2 and node 3 were blocked using the firewall, the average latency changed to 142.9 ms, due to the longer route that packets had to travel. Once we lifted the restriction on node 2, this average latency dropped back to 38.2 ms. Figure 6 shows that the cumulative number of packets received at the base station from node 2 remained unchanged during the period when the firewall was blocking all communications to and from node 2.

SOS does not permit two modules with the same module ID to exist in the system. As a result, to update a module, one has to remove it first and then

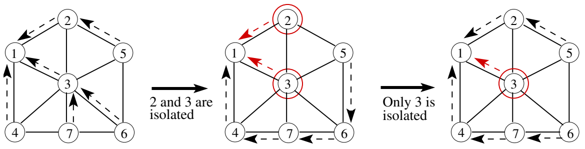


Fig. 5. Controlling Tree Routing in SOS using AEGIS. Communication from/to the circled nodes were blocked using firewall rules. Node 1 is the base station.

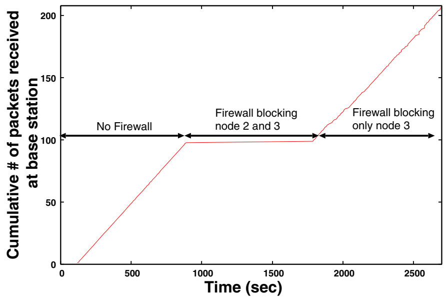


Fig. 6. Cumulative no. of packets received at the base station from node 2 in the topology shown in Figure 5

install the new version. If FModule itself is updated using this procedure, then there will be no firewall available during the update process, which is undesirable. To address this, we used two copies of FModule with different module IDs. While intercepting any communication, the kernel checked whether any one of them was available. The first available FModule was used to perform packet filtering. This way, we made sure that at least one FModule was always available. While updating FModule, we maintained the old copy until the the new version was installed. Once installed, we removed the old FModule. Thus, a firewall was always present in the system.

4.3 Creating Network Overlays

Aegis also allows packet filtering based on module IDs as opposed to node IDs. This capability enables Aegis to be used in a variety of interesting applications. Our final experiment, conducted using a small testbed of MicaZ nodes, demonstrates this. In this experiment, using simple firewall rules, we created two network overlays within the physical topology shown in Figure 7. Such overlays provide a way to create abstract topologies on top of a physical one. These abstractions can be used to reduce network traffic, run different applications on different subsets of nodes, *etc.,* all without altering the physical topology of the network .

In this experiment, we used two different versions of Surge, namely SurgeSlow (module ID 142) and SurgeFast (module ID 168). They collected data at

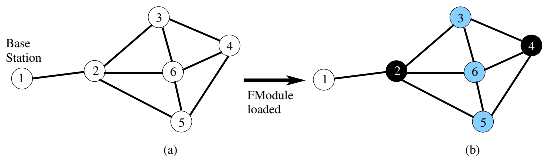


Fig. 7. Creating two network overlays using Aegis. Overlay 1 (nodes 2 and 4) was forwarding Surge data at a slower rate while overlay 2 (nodes 3, 5 and 6) was forwarding Surge data with a faster rate.

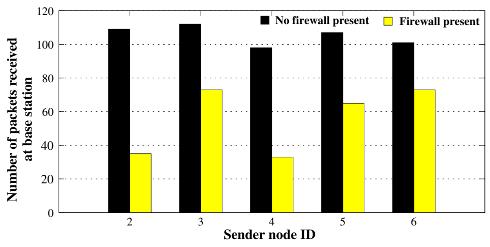


Fig. 8. Packet delivery statistics of the nodes from the two overlays in Figure 7

a rate of one sample per 8 seconds and one sample per 4 seconds respectively. Our goal was to run SurgeSlow on nodes 2 and 4 and SurgeFast on nodes 3, 5 and 6 (Figure 7). The ideal way to do this would be to install SurgeSlow and SurgeFast to the corresponding nodes only. But, as mentioned earlier, the module distribution mechanism in SOS does not provide the ability to dynamically dispatch a module to a specific node or set of nodes. Therefore, in our MicaZ testbed, all 6 motes were running both versions of Surge as shown in Figure 7(a). We loaded FModule to the network which was generated from these four rules: `M(2, 4) 168 OUT * DENY`, `* OUT M(2, 4) 168 DENY`, `M(3, 5, 6) 142 OUT * DENY` and `* OUT M(3, 5, 6) 142 DENY`. These rules allowed SurgeSlow data from only nodes 2 and 4 (Overlay 1) to flow towards the base station. Similarly, SurgeFast data from only nodes 3, 5 and 6 (Overlay 2) were permitted to flow towards the base station. Note that no incoming packets were blocked by the firewall. The reason was that Surge packets were wrapped inside TreeRouting (module ID 141) packets while traversing from one node to another node. As can be seen in Figure 8, nodes from overlay 2 were generating data at a rate about twice as fast as nodes from overlay 1. The data was collected for 300 seconds once TreeRouting stabilized. The results shown are averages over three runs.

This concept of creating network overlays can potentially be used in shared sensor network testbeds such as MoteLab [23], where our solution will provide the ability for multiple users to use the testbed simultaneously, providing a better utilization of shared resources.

5 Related Work

S_Firewall [19] provides defense against intrusion detection through the use of *mobile monitor agents*. These agents reside in a node to monitor the behavior of its neighbors. S_Firewall is not a rule based firewall like AEGIS and hence does not provide a true firewall experience. It is only focused on the intrusion detection and response problem. Murthy *et al.* [20] proposed a firewall solution where the firewall protects a fixed wired network that allows wireless users to connect to it. Their solution is designed for general wireless networks and did not consider the resource limitations of sensor networks. Moreover, individual wireless devices do not have a firewall running and hence they can not exploit the true benefits of a firewall. Hermes [14] is a software architecture that achieves visibility and control in WSN by interposing individual modules. In Section 3.1, we described why Hermes can not provide an effective firewall solution for WSNs. Use of virtual machines [16] to build a firewall is also not feasible due to its extra overheads.

Despite the lack of a firewall solution, security in WSN has been studied extensively [21]. Solutions for secure reprogramming [7], secure broadcasting [13], DoS attacks [25] and various other types of attacks were proposed. A firewall, like Aegis, can complement many of these solutions and thus eliminate some of the overheads associated with them, if not their necessity. In short, a well designed firewall can provide security against a number of attacks, access control mechanism and an effective intrusion prevention system (IPS) to low powered sensor networks, all under the same umbrella.

6 Conclusion and Future Directions

Prior experience has taught us that firewalls play a crucial role in safeguarding networked PC systems against remotely-launched security attacks. As WSNs evolve towards being connected to the Internet with the development of technologies such as 6LoWPAN, it becomes crucial to extend the protection offered by a firewall to this class of systems. As a first step, this paper presented Aegis, a lightweight, rule-based firewall for WSNs. Aegis has been designed to be resource-efficient, flexible, and easy to use. We have implemented Aegis using the SOS operating system and evaluated it through experiments conducted on real sensor nodes. Our results demonstrate that Aegis successfully performs gatekeeping of a sensor node's communication traffic in a flexible manner with minimal overheads. However, scope for future enhancements is still wide open. First, similar to all packet-filtering firewalls that filter packets based on node identifiers (*e.g.,* node ID, IP address), Aegis is vulnerable to attacks where the node identifier is faked (*e.g.,* IP spoofing attack). Node authentication schemes can be used to defend against such spoofing attacks. Second, despite the resource limitations that exist in WSNs, a stateful packet filtering firewall will provide more security and control. Introducing stateful behavior in Aegis will be one of our future goals. Third, extending Aegis to control not just whether packets are permitted to pass through, but also the amount of incoming/outgoing traffic per unit time will allow it to be used as a network resource manager. Fourth, in the current implementation of Aegis, synchronous communication [10] between modules on the same node can still bypass Aegis. This can be addressed by using Aegis in conjuction with a system such as Hermes [14].

References

1. TinyOS, `http://www.tinyos.net`
2. Bhatti, S., et al.: MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. Mobile Networks and Applications 10(4), 563–579 (2005)

3. Buonadonna, P., Hill, J., Culler, D.: Active message communication for tiny networked sensors. In: Proc. of INFOCOM (2001)
4. Chapman, D.B., Zwicky, E.D., Russell, D.: Building internet firewalls. O'Reilly & Associates, Inc., Sebastopol (1995)
5. Dunkels, A.: Full TCP/IP for 8 Bit Architectures. Proc. of MobiSys (May 2003)
6. Dunkels, A., Gronvall, B., Voigt, T.: Contiki-a lightweight and flexible operating system for tiny networked sensors. In: Proc. of the First IEEE Workshop on Embedded Networked Sensors, pp. 455–462 (2004)
7. Dutta, P., Hui, J., Chu, D., Culler, D.: Securing the deluge Network programming system. In: Proc. of IPSN, pp. 326–333 (2006)
8. Gershenfeld, N., Krikorian, R., Cohen, D.: The Internet of Things. Scientific American 291(4), 76–81 (2004)
9. Gouda, M.G., Liu, X.-Y.A.: Firewall design: consistency, completeness, and compactness. In: Proc. of 24th International Conference on Distributed Computing Systems, pp. 320–327 (2004)
10. Han, C.C., Rengaswamy, R.K., Shea, R., Kohler, E., Srivastava, M.: SOS: A dynamic operating system for sensor networks. In: MobiSys, pp. 163–176 (2005)
11. Hui, J.W., Culler, D.E.: IP is dead, long live IP for wireless sensor networks. In: Proc. of SenSys, pp. 15–28 (2008)
12. Hui, J.W., Culler, D.E.: Extending IP to low-power, wireless personal area networks. IEEE Internet Computing, 37–45 (2008)
13. Karlof, C., Sastry, N., Wagner, D.: TinySec: a link layer security architecture for wireless sensor networks. In: Proc. of SenSys, pp. 162–175 (2004)
14. Kothari, N., Nagaraja, K., Raghunathan, V., Sultan, F., Chakradhar, S.: HERMES: A Software Architecture for Visibility and Control in Wireless Sensor Network Deployments. In: IPSN, pp. 395–406 (2008)
15. Kumar, R., Kohler, E., Srivastava, M.: Harbor: software-based memory protection for sensor nodes. In: Proc. of IPSN, pp. 340–349 (2007)
16. Levis, P., Culler, D.: Mate: A Tiny Virtual Machine for Sensor Networks. In: Proc. of ASPLOS (2002)
17. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proc. of NSDI, vol. 246 (2004)
18. Liu, A.X., Torng, E., Meiners, C.R.: Firewall compressor: An algorithm for minimizing firewall policies. In: INFOCOM, April 2008, pp. 176–180 (2008)
19. Ma, J., et al.: S_Firewall: A Firewall in Wireless Sensor Networks. In: WiCOM, September 2006, pp. 1–4 (2006)
20. Murthy, U., Bukhres, O., Winn, W., Vanderdez, E.: Firewalls for security in wireless networks. In: Proc. of HICSS, vol. 7, p. 672 (1998)
21. Perrig, A., Stankovic, J., Wagner, D.: Security in wireless sensor networks. Commun. ACM 47(6), 53–57 (2004)
22. Titzer, B.L., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: IPSN, pp. 477–482 (April 2005)
23. Werner-Allen, G., Swieskowski, P., Welsh, M.: Motelab: a wireless sensor network testbed. In: Proc. of IPSN, pp. 483–488 (2005)
24. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proc. of SenSys, pp. 14–27 (2003)
25. Wood, A.D., Stankovic, J.A.: Denial of service in sensor networks. Computer, 54–62 (2002)

Halo: Managing Node Rendezvous in Opportunistic Sensor Networks

Shane B. Eisenman[1], Hong Lu[2], and Andrew T. Campbell[2]

[1] Columbia University, New York NY 10027, USA
shane@ee.columbia.edu
[2] Dartmouth College, Hanover NH 03755, USA
campbell@cs.dartmouth.edu

Abstract. One vision of an opportunistic sensor network (OSN) uses sensor access points (SAPs) to assign mobile sensors with sensing tasks submitted by applications that could be running anywhere. Tasked mobile sensors might upload sensed data back to these applications via subsequent encounters with this SAP tier. In a people-centric OSN, node mobility is uncontrolled and the architecture relies on opportunistic rendezvous between human-carried sensors and SAPs to provide tasking/uploading opportunities. However, in many reasonable scenarios application queries have a degree of time sensitivity such that the sensing target must be sampled and/or the resulting sensed data must be uploaded within a certain time window to be of greatest value. *Halo* efficiently, in terms of packet overhead and mobile sensor energy, provides improved delay performance in OSNs by: (i) managing tasking/uploading opportunity, and (ii) using mobility-informed scheduling at the SAP.

1 Introduction

The initial application focus of wireless sensor networking has been on *in situ* monitoring of ecological processes, or on industrial processes and equipment. Recently we and other researchers in the field have begun to consider the urban domain with a focus on *people-centric* [14] [16] [4] [22] sensing and application development. Architectures in this new domain assume mobile smart phones and embedded sensing devices equipped with a short-range radio (*e.g.*, ZigBee, Bluetooth, WiFi) are carried by humans or mounted on vehicles, leading to a network of sensors with mobility uncontrolled by the sensing architecture [11] [5]. Such architectures often employ a multi-tiered hierarchical structure where sensor tasking (*i.e.*, application query assignment) and data collection occur via mobility-enabled interactions between people-centric *mobile sensors (MSs)*, and edge wireless access nodes [10] we call *sensor access points (SAPs)*. In this context, we treat the question of how we may best task MSs and collect data from MSs in support of delay-aware applications.

Tasking and collection operations can occur when MSs enter the "spheres of interaction" of the SAPs. Generally, by sphere of interaction we mean the region (*i.e.*, the physical volume) within which services offered by a node are available to its neighbors. For the SAP case, to which we limit our discussion in this paper, these services include tasking and uploading. In practice, the adaptation of the sphere of interaction is implemented by both transmit power control, and multihop signaling between a SAP and MSs that happen to dwell for a time near a SAP. During their stay these may be used to funnel packets from data collecting MSs to the SAP, and to relay tasking messages from the SAP to MSs.

While applications that use opportunistic sensor networks should be delay tolerant, we draw a distinction between those that are delay-aware and those that are not. Delay-aware applications do not warrant real-time treatment, but may issue queries that have a degree of time sensitivity such that the sensing target must be sampled, and/or the resulting data must be uploaded, within a specified time window to be of greatest value. Examples are myriad, and include personal applications that seek to answer questions like, “Where can I find a quiet place to study for the next hour”, and public utility applications that say, “Give me my local weather spotter data in time for the next newscast”. In support of delay-aware applications, we investigate a number of fundamental performance issues in OSNs, including the interplay between resource consumption and the timeliness of tasking and data collection.

To increase the frequency and duration of the sensors’ travel through the sphere of interaction of a given SAP, the SAP might enlarge its sphere of interaction by increasing its transmission power and/or by building a multi-hop sphere of interaction. However, a multi-hop sphere of interaction requires increased signaling (*e.g.*, to set up and maintain routes), requiring more energy expenditure. In a setting with uniform per-link loss probabilities and fixed-power transceivers, multi-hop communication also implies a higher end-to-end loss probability compared to that possible; in a wireless environment with a link packet loss rate p the probability of success across n hops is $(1-p)^n$. Further, increasing the transmission power of the SAP implies an increased energy drain on the energy-limited MSs since these must match the higher transmission power of the SAP for bidirectional communications. Finally, a larger SAP sphere of interaction disrupts local peer-to-peer sensor communication in a larger part of the field, a problem of increasing relevance given the recent interest in mobile peer-to-peer services using localized communication [12] [20] [15].

We design, implement and evaluate Halo, a framework providing algorithmic and protocol support for managing rendezvous between SAPs and human-carried and vehicle-mounted mobile sensors in the urban domain. Halo manages opportunities for tasking and uploading operations via deadline-driven adaptation of SAP sphere of interaction. When multiple simultaneous operations are possible, Halo takes a snapshot of the system (*i.e.*, the sensors available for tasking and uploading in its sphere of interaction, the pending tasking operations, and the applications waiting for data upload), and incorporates sensor-driven mobility prediction of the available MSs to generate a schedule of the tasking and uploading operations. This novel scheduling approach integrates a traditional shortest-job-first approach, and a mobility-based approach tailored for OSNs.

2 Managing Rendezvous Opportunity

There are competing pressures in managing tasking and collection opportunities. We wish to expand the SAP sphere of interaction to increase the number of MSs available to task, reduce tasking delay, increase the amount and utility of sensed data to delay-aware applications, reduce collection delay, and reduce the likelihood of mobile sensor storage overflow. On the other hand, we wish to contract sphere of interaction to reduce required energy expenditure of mobile sensors when transmitting to a SAP, reduce disruption to communications ongoing between MSs in the vicinity of a SAP, and increase the security of the system by probabilistically reducing overhearing, and explicitly limiting the number of nodes offering (authenticated) proxy service on behalf of the SAP.

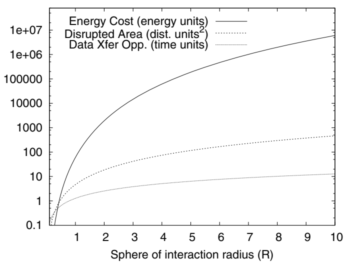


Fig. 1. Impact of SAP sphere of interaction radius

We use a simple model of a single SAP to illustrate the impact of sphere of interaction radius on data transfer opportunity between MS and SAP, required MS transmit energy, and the SAP interference area. Here, the radius of the "sphere" (our 2D analysis can directly be extended to 3D) is a real valued abstraction of the range extension due to power control only. The trigonometry is straightforward and is omitted due to space constraints, but details are available in the technical report [8, Appx I]. Figure 1 shows: (i) the data transfer opportunity (in abstract time units) of a single MS by plotting the average straight line trajectory length through the SAP sphere of interaction, assuming the MS maintains a constant unit velocity along the trajectory; (ii) the average transmit energy a MS must use to communicate with the SAP, assuming a symmetric link and a simplified Friis model with a loss exponent of 4, as it traverses the sphere of interaction along the average length chord; and (iii) the lower bound on the area disrupted by SAP-MS communications as the MS travels along the average length chord. Data transfer opportunity grows linearly with the sphere of interaction radius, while energy cost and SAP interference experience super-quadratic growth. The tradeoff between data transfer opportunity, and MS energy and SAP interference impact, motivates a managed SAP sphere of interaction radius. This illustrative analysis necessarily ignores important realities of wireless networks: the relationship between transmission power and physical distance between the SAP and MS is complex due to antenna characteristics, the attenuation by the body [13] and other environmental factors, and humans typically do not walk along chords. However, we reasonably assume that increasing SAP transmission power increases the probability of interaction between SAPs and MSs, and insofar as this happens the disrupted area and energy costs to the network also increase. Section 4 provides simulation results bolstering these numeric arguments.

While there are a number of possible triggers for increasing or decreasing the SAP sphere of interaction, we believe the fundamental driver for sphere of interaction adaptation should be fulfilling application requests (*i.e.*, tasking and collection operations) since this is the metric that mostly closely reflects the user experience. Generally, we wish to expand the sphere of interaction when application demands require it, and contract the sphere of interaction at all other times to reduce energy consumption and channel access contention in the vicinity of the SAP. In an OSN architecture, the baseline is the lazy approach where we passively wait on mobility to bring suitable sensors to task, or previously tasked sensors carrying back sensed data within the radio range of the SAP when the transmit power is fixed. However, if some sensed data are most valuable if sensed and/or delivered within a particular time window, an improvement over the performance of this lazy approach is required.

Assume we have an application query i with which to task the sensor network that requires data from a particular sensor type (*e.g.*, CO_2 sensor). Suppose that the data must be sensed at time $t_{s_i}(min) \leq t \leq t_{s_i}(max)$ to capture the event of interest (*e.g.*, rush hour pollution), and that a constant T_i exists that reflects the time it takes to travel from the tasking SAP (which is assumed to know its location) and the sensing target location defined in the query, using average case human speed. When an application query is inserted into the SAP task queue at time t_i^0, we calculate the time until sphere expansion $\delta_{s_i}^0 = (t_{s(max)_i} - t_i^0) - T_i$. If a MS matching the task requirements is available for tasking within the current SAP sphere of interaction, then no sphere adjustment is necessary and the tasking operation can proceed. Otherwise, at any time t^j a SAP calculates its sensing-driven sphere adjustment multiplier ξ_s for query i as $\xi_{s_i} = (1 - \delta_{s_i}^j / \delta_{s_i}^0)$.

Similarly, assume for an application query i, an MS was previously tasked and was able to sample the requested target. Suppose the data must be delivered back to a SAP by time $t \leq t_u$ in order for the data to have the greatest utility, and T_i is defined as before. Then at any time t^j a SAP calculates its upload-driven sphere adjustment multiplier ξ_u for query i as $\xi_{u_i} = (1 - \delta_{u_i}^j / \delta_{u_i}^{s(max)})$, where $\delta_{u_i}^{s(max)} = (t_{u_i} - t_i^{s(max)}) - T_i$. Queries not specifying sensing target locations set T_i=0. Queries not having sensing or uploading deadlines, set $t_s(max) = \infty$ and $t_u = \infty$, respectively.

We use a small set of power settings at both the MSs and SAPs, and limit the maximum number of hops of sphere expansion to keep the cost and complexity of interactions low. The choice of the supported power levels can be arbitrary or based on historical information kept at the SAP about the number of MSs found at a given sphere radius. Let $P = \{p_1, ..., p_K\}$ be the supported power levels at each node and let M be the maximum allowed number of hops. Then there are $M \cdot K$ possible sphere extension settings, and we write the set of settings as $S = \{s_1, ..., s_{M \cdot K}\}$, where for s_j, the number of hops $m = 1 + \lfloor \frac{j-1}{K} \rfloor$ and the power setting of the last hop $k = j \ mod(K+1)$ (hops prior to the last hop are at power setting p_K).

For a set of tasks Q at a particular SAP, the sphere of interaction is set according to

$$s = \max \left(\left\lceil \max_Q(\xi_{s_i}) \cdot M \cdot K \right\rceil, \left\lceil \max_Q(\xi_{u_i}) \cdot M \cdot K \right\rceil \right) \tag{1}$$

Following this rule, the sphere of interaction setting adapts to the current set of pending deadlines. Taking tasking as an example, as the time t^j gets closer to $t_{s(max)_i} - T_i$, then $\delta^j_{s_i}$ goes to 0 and the sphere setting grows to $M \cdot K$. On the other hand, if all pending deadlines are far enough in the future, then $\delta^j_{s_i}$ is still close to $\delta^0_{s_i}$ and the sphere setting shrinks close to the minimum. While a number of variations on this scheme are possible, the rule in Equation 1 has the advantage of encouraging early submission of application queries to the system. Though outside the scope of our current work, early submission might allow for query load balancing among SAPs, and smart assignment to particular SAPs.

3 Scheduling Operations

We focus on two scheduling design choices that impact both the efficiency of communications between the SAPs and MSs, the average operation turnaround time and the average operation throughput. First, we discuss reasons for serving a single MS until its operation is completed (or it leaves the SAP range) rather than switching between multiple MS sessions. Second we discuss how the SAP determines the order in which it will serve the MSs in its current sphere of interaction. To support scheduling, the SAP takes a snapshot of the system at particular points in time. Within this freeze frame, the SAP knows: the set of application tasks to complete, the set of MSs in the sphere of interaction of a SAP available for tasking, the set of MSs in the sphere of interaction of a SAP offering data to upload, the set of applications waiting for particular data, and an estimation of each node's proximity and mobility.

Atomic vs. Interleaved. Prior experimentation [4] with a testbed of Moteiv Tmote Invent motes (which use IEEE 802.15.4 radios) shows that at typical walking speeds and relatively low density of MSs, simultaneous uploading and tasking results in none of the operations being fully completed using state of the art sensor network transports. More recently, Miluzzo, *et al.* have characterized [13] the severe radio attenuation caused by the body that will be prevalent in human-centric networks, that will tend to limit the average contact time between SAP and MS even if higher data rate radios are used. Because of this limited contact time, an interleaved approach to operation scheduling (*i.e.*, either preemptive or simultaneous uploads and downloads) may not be appropriate. Firstly, a preemptive approach leads to a longer average turnaround time than non-preemptive scheduling [18] for all operations. Also, with single channel radios, simultaneous operations implies more MAC layer overhead in terms of either backoffs or collisions in the case of contention-based MACs, or schedule maintenance and dissemination in the case of non-contention-based MACs. Instead, we take a non-interleaved or atomic approach with at most one active uploading or tasking session ongoing at each SAP.

Scheduling Discipline. To determine the order in-sphere MSs will be served, a simple approach is to not actively manage the order of operations at all and just serve MSs in the order they arrive at the SAP until they move out of range. However, service disciplines like FIFO or even random selection ignore important features such as the size (*i.e.*, number of bytes) of tasking and uploading operations, and the MS dwell time in the SAP sphere of interaction. Thus, these naive approaches can lead to a lower operation throughput due to non-uniform MS inter-SAP-visitation times, hereafter *orbits*.

We propose a hybrid mobility-based/shortest-job-first (MB-SJF) scheduling algorithm to decide the order in which MSs are atomically served. Let A denote the event that a tasking or uploading operation supported by the current set of MSs can be completed before the chosen MS exits the maximum SAP sphere of interaction obtainable without multi-hop, since multi-hop extension is not always possible. Uploading and tasking operations are ordered by $Prob(A)$. This probability reflects the size of the operation to be completed (*i.e.*, number of bytes b) and the estimated dwell time, t_{SAP}, of the associated MS in the sphere of interaction. We have

$$Prob(A) = Prob(t_{SAP} \geq \frac{b}{C} + \beta), \tag{2}$$

where C is the wireless channel rate in bytes per second, and $\beta = \overline{BO} \cdot \frac{b}{frame\_size}$, for a CSMA channel. Here, $\overline{BO}$ is the average MAC backoff interval across the packets needed to complete the operation, and $frame\_size$ is the maximum MAC frame size in bytes. It is worth nothing that with our atomic scheduling approach, the second term on the right side of the inequality can be driven to zero if the MAC parameters are tweaked such that a backoff window of zero is used during an upload/download session between a MS and the SAP (the standard backoff window would still be used for communication between MSs and for the MS$\leftrightarrow$SAP session setup [8, Appx II]).

In practice an empirically-derived mean value estimate of an MS's SAP dwell time $\bar{t}_{SAP}$ is easily tracked by the MS and shared with the SAP each rendezvous. Cold start effects on $\bar{t}_{SAP}$ are mitigated by seeding with values averaged across the MS population. Since we are considering the people-centric sensing domain, human activity inferred from on-board sensors can aid the MS in further refining its $\bar{t}_{SAP}$ estimate. In particular, samples from an accelerometer (embedded in many new mobile phone devices) can be processed to determine if a person is standing, walking, or running. In [12], the authors classify between these three states with an average accuracy of about 90%. Since average dwell times are likely to be highly correlated with human activity, we propose to keep a separate $\bar{t}_{SAP}$ for each classified activity and use this value in calculating Equation 2.

Once $Prob(A)_i$ is calculated for each {operation, MS} pair i in the sphere of interaction, the operation schedule is set in descending order of the value $Prob(A)_i * \nu_i$. The optional priority factor ν_i can be used to prioritize certain event types (*e.g.*, toxic spill) or users (*e.g.*, those with long average orbits), but the exact meaning is left up to the system administrator and it may be dropped altogether if user/operation priority need not be supported.

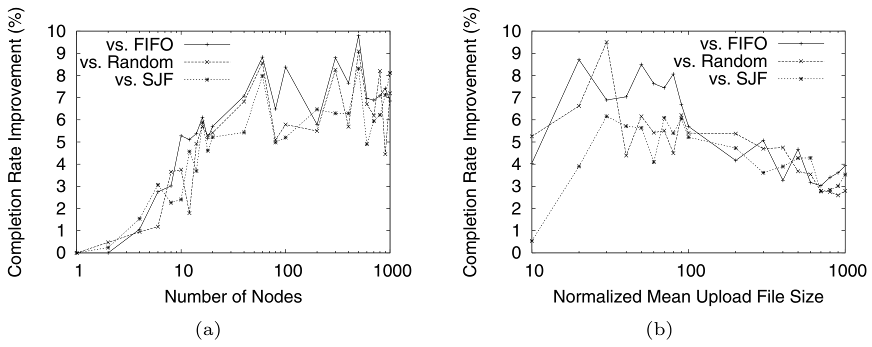


Fig. 2. Advantage of MB-SJF versus other common scheduling disciplines in simulation. MB-SJF consistently completes the upload tasks faster than FIFO, RAND and SJF, across all tested parameter values for node population and mean file size.

To evaluate the performance of MB-SJF, we simulate a one-SAP/multi-MS scenario where all MSs are assumed to have data to upload. We compare MB-SJF with common scheduling disciplines such as first-in-first-out (FIFO), random selection (RAND), and shortest-remaining-job-first (SJF) in terms of the time it takes to upload the data from all of the MSs. Each of the MSs is assigned a file to upload whose size is randomly chosen from an exponential distribution. MSs move between two states, at-SAP and not-at-SAP, where the dwell times (t_{SAP}) in each state are randomly drawn from different exponential distributions with means λ_{SAP} and $\lambda_{\overline{SAP}}$, respectively. To simulate a population of MSs with different mobility characteristics, each node is assigned a unique $\{\lambda_{SAP}, \lambda_{\overline{SAP}}\}$ pair, whose values are uniformly spread between 0 and $N\gamma$, where N is the number of MSs and γ is the spreading factor. The simulator updates MS states synchronously and both the file sizes and the location dwell times are normalized to its update period. For MB-SJF, it is assumed the MSs report their λ_{SAP} and $\lambda_{\overline{SAP}}$ values and remaining file size to upload (b) to the SAP upon entering the SAP's sphere of interaction (c.f., the *beacon reply* message in [8, Appx II]. Neglecting MAC effects, from Equation 2 the SAP computes $Prob(A) = e^{-\lambda_{SAP} b}$ for each node in its sphere. $\nu = e^{-\lambda_{\overline{SAP}}}$ prioritizes nodes with long orbits.

In Figures 2(a) and 2(b), we plot the completion rate improvement MB-SJF gives versus FIFO, RAND, and SJF. Each point represents the average of 1000 trials, each with a different seed for the pseudo-random number generator driving upload file sizes and location dwell times.

Figure 2(a) shows the completion rate improvement versus the number of MSs in the simulation, with a fixed mean upload file size of 100. As the MS population grows, the advantage of MB-SJF steadily increases until settling around a 6-10% improvement after 60 nodes. MB-SJF, like plain SJF, is able to finish off small upload tasks quickly, but also takes advantage of mobility information to opportunely upload from nodes that visit the SAP relatively rarely. Figure 2(b) shows the completion rate improvement versus the mean upload file size when there are 20 MSs. As expected, for medium size files (implying medium aggregate upload times), MB-SFJ provides for a relatively constant improvement in completion rate. As file sizes get larger, the improvement begins to diminish. As file sizes tend to infinity, so does the completion time regardless of scheduling discipline, and therefore the possible improvement goes to 0. However, we believe the typical case for opportunistic wireless uploads from mobile consumer devices will be small to medium size files (*e.g.*, a 1kB text file, a 1MB image file, a 10MB audio file). Based on these simulations, MB-SJF seems to be a good candidate for MS scheduling in the SAP sphere of interaction and we use MB-SJF for further evaluation in Section 4.

Scheduling Epoch. MSs may enter a SAP's sphere of interaction during an ongoing schedule. In this case, the SAP could ignore all newcomers until its current schedule is complete and then come up with a new schedule that incorporates the newcomers, or it might create a new schedule upon the completion of every operation. In the former case, starvation is prevented, but new sensors that may rank higher are ignored. In the latter case, more energy is spent and time wasted by re-running the neighbor discovery after every operation. In Halo, we define a scheduling epoch of time length E to strike a middle ground. E is adaptive to the estimated mobility of the MSs involved in the current schedule. Let K_i represent this set of MSs for a given schedule; then $E = \max_{K_i}(t_{SAP})$. The next scheduling time is then defined as

$$t_{sched_{i+1}} = t_{sched_i} + min\left(\sum_{j=1}^{n-1} \ell_j^{sched_i}, \sum_{j=1}^{|K|} \ell_j^{sched_i}\right), \qquad (3)$$

where $\ell_j^{sched_i}$ is the length of the jth operation in schedule i, and n is the ordinal of the first operation that makes the sum greater than E. MSs that depart the SAP's sphere of interaction before their schedule slot are skipped. This method of addressing starvation is more appropriate than standard aging techniques since MSs with a lower $P(A)$ (later in the schedule) are not likely to be around very long and would not be able to take advantage of the aging. Thus, with the schedule epoch, we focus on getting the higher $P(A)$ operations done rather than on fairness with respect to starvation.

4 Halo Evaluation

We base our evaluation of deadline-driven sphere of interaction management on the comparison of three schemes: MIN, ADAPT, and MAX. We use ten SAP sphere of interaction settings $S = \{s_1, ...s_{10}\}$, with $M = 1$ and $K = 10$, where setting s_j corresponds to a sphere radius of $j * 3$ distance units. This relationship implies a field without radio obstructions, which is unlikely in a people-centric network. We make this simplifying assumption to avoid making arbitrary assumptions about the deployment environment. Note, however, that transmission distance will always increase monotonically with transmission power. In the MIN scheme SAPs always use s_1; in the MAX scheme SAPs always use s_{10}; in the ADAPT scheme each SAP independently varies its radius according to the sensing deadlines of tasks it is managing. All schemes use MB-SJF scheduling.

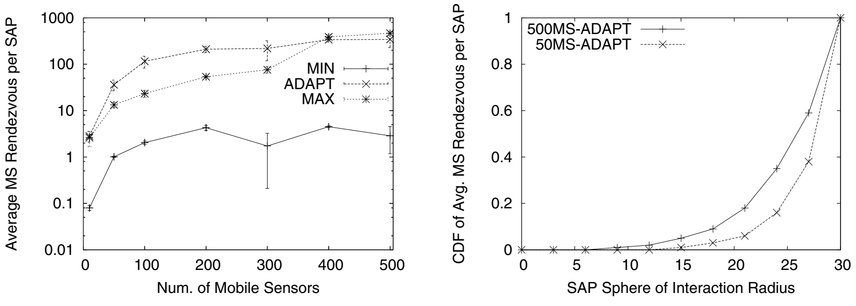
Simulation Environment. We implement Halo algorithms and the tasking and uploading communications protocols (see [8, Appx II] for details) in nesC, and simulate several multi-SAP multi-MS scenarios using TOSSIM/Tython. TOSSIM simulates Halo on the TinyOS platform, including packet exchange, timer events, etc. Tython is a Python/Java front end used to manage node mobility and connectivity.

Each simulation trial is conducted on a 500×500 field. MSs are initially placed uniformly at random across the field and move according to a modified random walk. MSs choose an activity uniformly at random from {standing, walking, running} and continue with that activity for a period of time chosen uniformly at random between 1 and 1200 seconds. According to the chosen activity MSs move at a rate of, respectively, {0, 3, 15} distance units per second in a direction chosen uniformly at random, between 1 and 360 degrees inclusive, at the same time the activity is chosen. MSs bounce off the field boundaries. 50 SAPs are placed uniformly across the field and remain stationary throughout the simulation.

MSs estimate their t_{SAP} and communicate this estimate to the SAP during the tasking exchange [8, Appx II] to facilitate the MB-SJF schedule calculation (see Equation 2). We assume all MSs have an accelerometer that can be used for activity classification, and use the activity classifier confusion matrix published in CenceMe [12, Fig 6(a)] to drive the actual and reported mobility characteristics in the simulation. For example, a MS may be "running" as dictated by the mobility model, but the MS believes it is running with only 90.9% probability. With 8.37% probability it thinks it is walking and tells the SAP the wrong information. Real world effects such as classification inaccuracy (or GPS error if a GPS system is used as a basis for a dwell time estimate) degrade the performance of the MB-SJF scheduler. Yet, even under worst case classification accuracy, the scheduler just behaves as a random scheduler that does not consider mobility.

As no large-scale mobile sensor networks allowing external queries have yet been built, we make some best guesses at parameters characterizing the task arrival process. Emulating application requests from thousands of backend system users, tasks arrive independently at each SAP with inter-arrival times drawn randomly from an exponential distribution with a mean of 10s. Task sizes are drawn randomly from an exponential distribution with a mean of 10 packets (packets are 128 bytes long). The sensing deadline ($t_{s(max)}$) is randomly chosen for each task from an exponential distribution with a mean of 1000s. The deadline threshold T should reflect the time it takes on average to travel the distance from the tasking SAP to the sensing target (see Section 2). In our simulation, the T value for a given task is chosen uniformly at random in the interval from 1 to $t_{s(max)}$. This is equivalent to choosing a sensing target uniformly at random in the field and pre-filtering those tasks whose sensing deadlines do not allow enough travel time from SAP to target. A task whose sensing deadline passes before it is assigned to a MS is dropped from the SAP's task queue. The SAP's *beacon* interval [8, Appx II] is set to 5s.

Each MS has a task queue of size 1, meaning it can only serve one application query at a time. If a task is partially transferred to a MS before a particular



(a) Impact of MS density on the number of MS/SAP rendezvous.

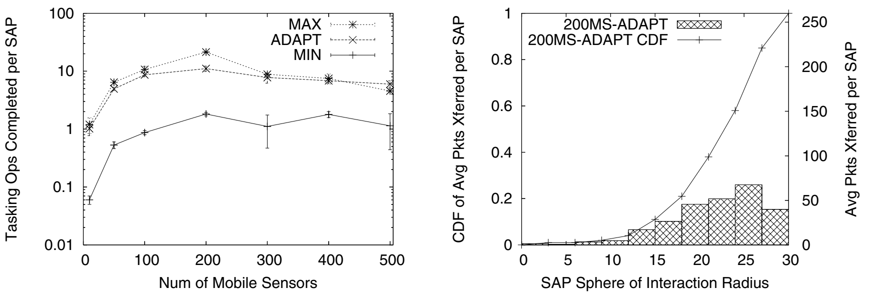
(b) Cumulative distribution of rendezvous vs. sphere radius for diffent MS densities (with ADAPT scheme).

Fig. 3.

tasking session completes, the MS caches the state of the suspended session and resumes the session at the next met SAP. If the sensing deadline of a task that is partially transferred to a MS expires, the partial state is expunged from the MS. MSs that are fully tasked and then successfully sense the target generate a number of data packets to upload chosen randomly from an exponential distribution with a mean of 100 packets. About 20% of the fully tasked MSs end up reaching their respective target sensing regions prior to their sensing deadlines. We use an uploading deadline of infinity for all tasks; a MS with data to upload maintains the state of its upload session across how ever many SAP rendezvous it takes to complete the upload.

In the following graphs, each data point represents the average of five one-hour trials, and error bars indicate the 95% confidence interval.

Impact on Tasking/Uploading Opportunity. To characterize the impact of sphere of interaction radius on the opportunity for MS/SAP rendezvous, we run simulations across a range of MS densities. Results are summarized in Figure 3. In Figure 3(a), we quantify the impact of MS density on the number of MS/SAP rendezvous, plotting the average number of MS/SAP rendezvous per SAP versus the number of MSs. The y-axis is in log scale to better show the detail despite the wide spread between MIN and MAX. Unsurprisingly, the number of rendezvous generally increases with increasing MS density for MIN, ADAPT and MAX. Of interest, the ADAPT scheme actually results in more rendezvous for the intermediate MS densities tested. This is likely due to the dynamism of the sphere of interaction, resulting in "re-rendezvousing" for MSs that have moved little (*e.g.*, standing) during the sphere adaptation time scale. The effect becomes negligible at the lowest densities (10 mobile sensors) since the overall probability of rendezvous shrinks dramatically. At high densities (*i.e.*, above 400 MSs), this effect is overwhelmed by the sheer number of mobility-based rendezvous, and at the highest density (500 MSs) MAX yields more rendezvous than ADAPT since it always uses the largest sphere radius. Another way to see the effect of MSs density is to consider the sphere of interaction radius at which most rendezvous



(a) MIN completes an order of magnitude fewer tasking ops than ADAPT or MAX, giving poor service to apps.

(b) Packets xferred generally increases with radius, but the largest radius shows a diminishing return.

Fig. 4.

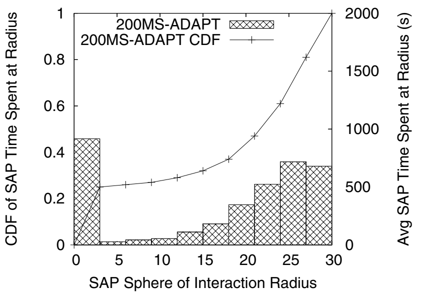
occur. In Figure 3(b), we show the cumulative distribution function (CDF) of the average number of MS rendezvous per SAP versus SAP sphere of interaction radius. Curves for 500 MSs and 50 MSs show how at lower densities the majority of rendezvous occur at higher values of sphere radius. For example, at a sphere radius of 27 (second largest), in the 50 MS scenario only 40% of the rendezvous have occurred, while in the 500 MS scenario already 60% have occurred.

Impact on Bytes Transferred and Operations Completed. Figure 4 summarizes the performance of the ADAPT scheme in terms of number of task and upload packets transferred between MSs and SAPs, and the number of tasking operations completed. Simulations are run across a range of MS densities.

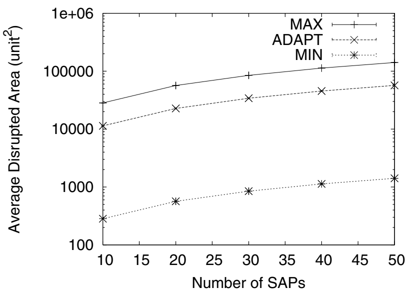
Figure 4(a) shows the average number of tasking operations completed per SAP plotted in log scale across a range of MS densities. We see that the behavior of adapting the sphere of interaction radius based on proximity to the sensing-deadline for a particular task leads to excellent comparative performance for ADAPT. On average ADAPT completes 85.5% of the tasks MAX does and nearly 10 times as many as MIN does.

In Figure 4(b), we provide insight into why the operation completion performance of ADAPT is able to remain close to that of MAX. Figure 4(b) shows the distribution (on the right axis) and cumulative distribution function (on the left axis) of packets transferred across sphere of interaction radius for the median density scenario (200 MSs). We see that, in contrast to the rendezvous distribution shown in Figure 3(b), the packet transfer distribution does not monotonically increase with increasing sphere radius. Rather, the amount of additional packets transferred at the maximum sphere radius is less than the penultimate radius, indicating a diminishing return for increasing the sphere radius.

Impact on Disrupted Area. Figure 5 summarizes the extent to which an increased sphere radius impacts the disrupted area. Since MS energy depletion is proportional to the sphere of interaction radius, we omit explicit energy-related results here. Figure 5(a) shows the distribution (and CDF) of the time that SAPs



(a) For ADAPT, SAPs spend a plurality of their time at the lowest setting, but the majority at the highest settings.



(b) Impact of SAP density on disrupted area. On avg., ADAPT disrupts $\approx \frac{2}{3}$ the area that MAX does.

Fig. 5.

on average spend at each of the 10 sphere of interaction settings when using the ADAPT scheme. The data are from the median density 200 MS scenario, but are similar for the other tested MS densities. SAPs spend a plurality of their time (about 25%) at the lowest sphere setting, implying the minimum possible disruption. Yet, in aggregate most of the time is spent at or above the eighth setting. In fact, the average sphere radius is about two thirds of that used by the MAX scheme. Figure 5(b) reflects this relationship and also indicates that the disturbed area in the field can quickly get very large as the SAP density increases (number of MSs is fixed at 200). The MIN scheme disrupts a much smaller area, but as we see in Figure 4 MIN also transfers and completes tasking operations at a rate an order of magnitude lower rate than ADAPT.

We define a unified efficiency metric as the average number of operations completed per unit area disrupted $\eta = Ops/Area$. On average across our tested desities $\eta_{\text{ADAPT}}/\eta_{\text{MIN}} = 0.27$ and $\eta_{\text{ADAPT}}/\eta_{\text{MAX}} = 2.18$; ADAPT gives a 200% improvement over MAX., while facilitating a nearly $10\times$ improvement over MIN in terms of completed operations. While η provides a notion of efficiency, it also reflects the tradeoff between resource conservation, *i.e.*, disrupted area and MS energy, and a coarse-grained quality of service in terms of system responsiveness to application queries.

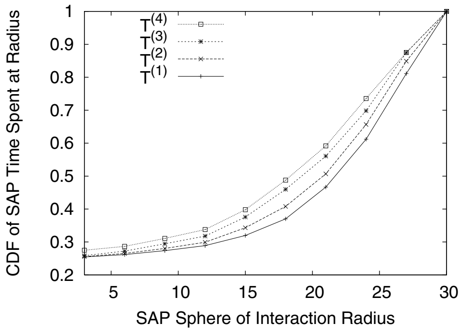


Fig. 6. As the deadline threshold T increases, the SAPs spend proportionally more time at higher sphere of interaction settings

In Figure 6, we illustrate how adjusting the deadline threshold T can be used to move the operating point of the ADAPT scheme from more resource conserving to offering a lower average completion delay to application queries.

For the 200-MS scenario, Figure 6 shows the CDF of the time that SAPs on average spend at each of the 10 sphere of interaction settings when using the ADAPT scheme for different values of the average deadline threshold, $\overline{T}$. In the previous simulations, T is chosen uniformly between 1 and $t_{s(max)}$, so $\overline{T} = t_{s(max)}/2$. In Figure 6, we adopt the following shorthand: $T^{(i)} = t_{s(max)}/2 \cdot i$. As $\overline{T}$ increases, SAPs spend proportionally more time at higher sphere settings, resulting in more packets transferred and better delay service to the applications but consuming more resources of the MS cloud (*i.e.*, energy and peer-to-peer communications opportunities).

Discussion. The MIN scheme provides the lowest energy consumption and disrupted area; the MAX scheme provides the greatest opportunity for packet transfer between MSs and SAPs, and thus the highest operation completion rate. In the previous sections we show that by dynamically adjusting the sphere of interaction, *i.e.*, transmit power and hop extension, according to the sensing deadlines of submitted application tasks ADAPT hits the sweet spot and provides increased opportunities for packet transfer and operation completion compared with MIN, and at a lower cost in terms of disruption to MS P2P communication compared to MAX. While we do not present explicit MS energy results, the packet reception energy is directly related to the disrupted area (see Figure 5(b)) and the packet transmission energy is directly related to the SAPs' transmit power setting (see Figure 5(a)). MS energy consumption and disrupted area can be considered as system design parameters based, for example, on MS battery characteristics and expected P2P application traffic in the mobile cloud, to drive the selection of the deadline threshold T.

5 Related Work

Managing the SAP sphere of interaction is related to adaptive clustering. Work from the MANET community proposes various clustering techniques, but invariably to increase routing efficiency and/or reduce routing protocol overhead (*e.g.*, [9] [7] [6] [17]). Zone Routing Protocol [9] sets a zone boundary between proactive and reactive routing to reduce the number of route request packets while providing good route acquisition delay. However, a method of determining an appropriate zone radius is not specified.

A study of adaptive clustering with the end of maximizing operation completion rate does not exist. The relationship between node density, transmission power, and neighbor set cardinality has been studied in the context of wireless graph connectivity [1] [23]. The effects of the relationships between transmission power, node density, and node mobility patterns on the operation completion rate have not been reported. A number of existing scheduling policies may be appropriate for SAP operation scheduling, but none have been evaluated with the unique combination of constraints present in a large scale mobile sensor network, to the best of our knowledge.

Halo's adaptation of a SAP's sphere of interaction is analogous to the "cell breathing" approach used in cellular telephony and proposed [3] for 802.11

access nodes for system load balancing. With a similar aim, the authors of the SoftRepeater [2] system use a combination of network coding and channel utilization to decide when 802.11 AP clients should become repeaters for others' traffic in order to address the "rate anomaly problem". Rather than dealing with SAP overloading, we address what is in some ways the opposite problem of application starvation due to under-utilization of the SAP tier.

Power control in cellular systems aims to save handset energy and secondarily to reduce adjacent cell interference. While we share the first concern, cellular mechanisms are too complex, and use a separate control channel which is not generally available on embedded sensing platforms.

6 Conclusion

We have shown how by adapting SAPs' spheres of interaction Halo can manage the opportunity for interaction between mobile sensors and sensor access points, while striking a balance between resource consumption and operation completion. Halo uses a new scheduling discipline (MB-SJF), based on mobility statistics and sensor-based inputs, that is tailored for the typical characteristics of the people-centric sensing domain. MB-SJF is shown to provide up to a 10% increase in operation completion rate compared to FIFO, random selection, and shortest-job-first scheduling, independent of the gains achievable from SAP sphere of interaction management. Halo provides improved support for delay-aware applications in the people-centric sensing context.

Acknowledgment

This work is supported in part by Intel Corp., Nokia, NSF NCS-0631289, and the Institute for Security Technology Studies (ISTS) at Dartmouth College. ISTS support is provided by the U.S. Dept. of Homeland Security under Grant Award Number 2006-CS-001-000001 and by award 60NANB6D6130 from the U.S. Dept. of Commerce. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Dept. of Homeland Security.

References

1. Agarwal, A., Kumar, P.R.: Capacity bounds for ad-hoc and hybrid wireless networks. ACM SIGCOMM CCR, Special Issue on Science of Networking Design 34(3), 71–81 (2004)
2. Bahl, P., Chandra, R., Lee, P.-C., Misra, V., Padhye, J., Rubenstein, D., Yu, Y.: Opportunistic Use of Client Repeaters to Improve Performance of WLANs. In: Proc. of ACM CoNEXT 2008, Madrid (December 2008)
3. Bahl, P., Hajiaghayi, M., Jain, K., Mirrokni, V., Qiu, L., Saberi, A.: Cell Breathing in Wireless LANs: Algorithms and Evaluation. IEEE Trans. on Mobile Computing 6(2) (February 2007)

4. Campbell, A., Eisenman, S., Lane, N., Miluzzo, E., Peterson, R.: People-Centric Urban Sensing. In: Proc. WICON 2006, Boston (August 2006)
5. Chaintreau, A., Mtibaa, A., Massoulie, L., Diot, C.: The Diameter of Opportunistic Mobile Networks. In: Proc. of ACM CoNEXT 2007, New York (December 2007)
6. Chatterjee, M., Das, S.K., Targut, D.: WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks. Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks) 5 (April 2002)
7. Du, S., et al.: Self-Organizing Hierarchical Routing for Scalable Ad Hoc Networking. ACM Ad Hoc Networks 6(4) (June 2008)
8. Eisenman, S.B., Campbell, A.T.: Managing Node Rendezvous is Opportunistic Sensor Networks. Tech. Report, http://www.ee.columbia.edu/~shane/halo.pdf
9. Haas, Z.J.: A New Routing Protocol for the Reconfigurable Wireless Networks. In: Proc. ICUPC 1997 (October 1997)
10. Hull, B., et al.: CarTel: A Distributed Mobile Sensor Computing System. In: Proc. of 4th ACM Int'l Conf. on Embedded Networked Sensor Systems, Boulder, pp. 125–138 (November 2006)
11. Laibowitz, M., Paradiso, J.A.: Parasitic Mobility for Pervasive Sensor Networks. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) PERVASIVE 2005. LNCS, vol. 3468, pp. 255–278. Springer, Heidelberg (2005)
12. Miluzzo, E., Lane, N.D., Eisenman, S.B., Campbell, A.T.: CenceMe - Injecting Sensing Presence into Social Networking Applications. In: Kortuem, G., Finney, J., Lea, R., Sundramoorthy, V. (eds.) EuroSSC 2007. LNCS, vol. 4793, pp. 1–28. Springer, Heidelberg (2007)
13. Miluzzo, E., Zheng, X., Fodor, K., Campbell, A.T.: Radio Characterization of 802.15.4 and its Impact on the Design of Mobile Sensor Networks. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913, pp. 171–188. Springer, Heidelberg (2008)
14. Parker, A., Reddy, S., Schmid, T., Chang, K., Saurabh, G., Srivastava, M., Hansen, M., Burke, J., Estrin, D., Allman, M., Paxson, V.: Network System Challenges in Selective Sharing and Verification for Personal, Social and Urban-scale Sensing Applications. In: Proc. of HotNets-V, Irvine (November 2006)
15. Ravi, N., Stern, P., Desai, N., Iftode, L.: Accessing ubiquitous services using smart phones. In: Proc. PERVASIVE 2005, March 8-12 (2005)
16. Sensorplanet, http://www.sensorplanet.org/
17. Sharony, J.: A Mobile Radio Network Architecture with Dynamically Changing Topology Using Virtual Subnets. In: Proc. ICC 1996, vol. 2, pp. 807–812 (June 1996)
18. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating Systems Concepts, 7th edn. John Wiley & Sons, Inc., Chichester (2004)
19. Srinivasan, K., Levis, P.: RSSI is Under Appreciated. In: Proc. EMNETS 2006, Cambridge, MA (2006)
20. Srinivasan, S., Moghadam, A., Hong, S.G., Schulzrinne, H.G.: 7DS - Node Cooperation and Information Exchange in Mostly Disconnected Networks. In: Proc. ICC 2007, June 1 (2007)
21. Srivastava, M., et al.: Wireless Urban Sensing. In: CENS Tech. Report #65 (April 2006)
22. Tuulos, V., Scheible, J., Nyholm, H.: Combining Web, Mobile Phones and Public Displays in Large-Scale: Manhattan Story Mashup. In: LaMarca, A., Langheinrich, M., Truong, K.N. (eds.) Pervasive 2007. LNCS, vol. 4480, pp. 37–54. Springer, Heidelberg (2007)
23. Zhang, X., Maxemchuk, N.F.: The Effects of the Number of Neighbors in Multihop Wireless Networks. Int'l Journal of Wireless and Mobile Computing, Special Issue on Group Communications in Ad hoc Networks (to appear)

Optimal Data Gathering Paths and Energy Balance Mechanisms in Wireless Networks

Aubin Jarry[1], Pierre Leone[1,⋆], Sotiris Nikoletseas[2], and Jose Rolim[1]

[1] Computer Science Department, University of Geneva, Battelle Batiment A, route de Drize 7, 1227 Geneva , Switzerland
[2] University of Patras and Computer Technology Institute N. Kazantzaki Str 1, Patras University Campus, 26504 Rion, Patras Greece

Abstract. This paper studies the data gathering problem in wireless networks, where data generated at the nodes has to be collected at a single sink. We investigate the relationship between routing optimality and fair resource management. In particular, we prove that for energy balanced data propagation, Pareto optimal routing and flow maximization are equivalent, and also prove that flow maximization is equivalent to maximizing the network lifetime. We algebraically characterize the network structures in which energy balanced data flows are maximal. Moreover, we algebraically characterize communication links which are not used by an optimal flow. This leads to the characterization of minimal network structures supporting the maximal flows.

We note that energy balance, although implying global optimality, is a local property that can be computed efficiently and in a distributed manner. We suggest online distributed algorithms for energy balance in different optimal network structures and numerically show their stability in particular setting. We remark that although the results obtained in this paper have a direct consequence in energy saving for wireless networks they do not limit themselves to this type of networks neither to energy as a resource. As a matter of fact, the results are much more general and can be used for any type of network and different type of resources.

1 Introduction, Our Contribution and Related Work

In full generality, this paper addresses the question and impact of fairly allocating resources while routing messages in networks. Resources belong to the nodes composing the network and the constraints emerging from resource limitation concern the traffic handled by nodes. By fairly allocating the resources we mean that their use must be proportionally distributed among the nodes in accordance to the node's available resources. To exemplify, we consider the particular, important case where the resource is the total energy available at the nodes for transmitting

⋆ Contact author: Pierre Leone, E-mail: pierre.leone@unige.ch. Partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS).

data. We consider the *data gathering* problem, where the nodes generate data that has to be collected by a unique sink. In this setting, nodes have generally many choices for routing the data to the sink following a multiple-hop pattern. The energy consumption of a node depends on the particular costs of the links chosen for transmitting the data. Classically, we are interested in Pareto optimal routing schemes which are such that no node can decrease its energy consumption without increasing the energy consumption of others. Although Pareto optimality is classically used to solve multiobjective optimization problems, this criterion usually does not define a routing scheme uniquely. However, we show that if we consider energy-balanced routing schemes then Pareto optimal and maximal flows are equivalent. This result is relevant because energy-balance is a local characteristic of flows and is suitable to be efficiently and distributively computed. Moreover, we show that maximizing the flow of data is equivalent to maximizing the lifetime of the network.

Another novelty of the paper is to algebraically characterize network's structure such that energy-balanced flows of data are maximal. We call such networks *energy-balance optimal*. This result is based on the equivalence between maximal flow and Pareto optimal solution provided that the flow is energy-balanced. Moreover, we also consider communication graphs Γ which contain an energy-balance optimal subgraph Λ and which are such that the maximal energy-balance flow in Λ cannot be increased by adding edges in Γ to the communication graph. In this case, we define that Λ is *energy-balance optimal in* Γ. As an application we investigate a particular simple topology which is energy-balance optimal in the complete graph and two realistic energy-balance optimal network structures.

To conclude the paper, we suggest an algorithm to online and distributively balance the energy-consumption of the nodes on the top of energy-balance optimal network structures. Numerical validations show that the algorithm is stable in the sense that the difference between the maximal and minimal energy consumption is bounded. Although theoretical works have still to be conducted to theoretically understand the conditions ensuring the existence of energy-balance flows, this (partly) validates our assertion that the local character of energy-balance flows is sutiable for distributed online algorithms.

An important application of our work is the ability to maximize the flow (and also the network lifetime) in any particular communication graph by generating an energy balanced flow. This is an important generalization over previous work ([7,8,11,15,5]). Interestingly, our results here imply (a) energy balanced data propagation using only two transmission levels (i.e. either to one hop neighbors or to the sink directly) is optimal, since they maximize the flow (b) and, we show the conditions under which we can compute such an energy balanced data propagation pattern.

The energy balance problem is particularly relevant in smart building scenaria and applications, in order to maximize the lifetime of the already deployed sensor network and also (via the flow maximization) to accelerate the collection of sensory readings. Furthermore, balancing the energy dissipation among the nodes of the network also contributes to keeping the electromagnetic radiation levels low, since this way the network operates efficiently and excess wireless transmission ranges may not be neeeded".

Reviewing the complete litterature on energy-balance mechanism is not possible here. In the following we review important contributions and we refer to the references therein. We point out that to the knowledge of the authors the approach developped in this paper is original.

An important relevant work is that of [17] where the authors define the energy balance property and propose, analyze and evaluate an energy-optimal and energy balanced algorithm for sorting in wireless sensor networks. In particular, they consider a single- hop sensor network. A similar approach is used in [16] for the important problem of selection. Also, [19] proposes an energy-balanced allocation of a real-time application onto a single-hop cluster of homogeneous sensor nodes connected with multiple wireless channels. An epoch-based application consisting of a set of communicating tasks is considered. Each sensor node is equipped with discrete dynamic voltage scaling (DVS). The time and energy costs of both computation and communication activities are considered. Both an Integer Linear Programming (ILP) formulation and a polynomial time 3-phase heuristic are proposed. Our work extends the approaches above to the general case of a multi-hop network and for the (quite general) problem of propagating data to the sink.

In [14] a new metric, "energy-welfare", is proposed that captures both the average and balance of sensor's remaining energy, and a Maximum Energy Welfare Routing protocol is provided, which achieves energy efficiency and energy balance of sensor networks; the treatment is by simulations and, although good performance gains are achieved, no fundamental relations between balance and efficiency are provided. In [4] the authors propose a distributed energy balance clustering protocol, in which cluster heads are selected by a probability depending on the ratio between remaining energy of node and the average energy of network. The high initial and remaining energy nodes have more chances to be the cluster heads than the low energy nodes. [12] proposes unequal clustaring towards balance the energy dissipation in each cluster; an earlier paper on unequal clustering is [18]. Compared to these approaches, our work is more general (not related to clustering only).

2 Balancing the Flow and Maximizing the Lifetime of a Network

In this section, we first prove that maximizing the lifetime of a sensor network is equivalent to solving a max-flow problem (Proposition 1). We then define energy-balanced flows (Definition 1) and give sufficient conditions under which energy-balanced flows are optimal (Proposition 2).

We consider a finite set of nodes and label them $i = 1, \ldots, N$. Nodes are able to communicate between each other only if they share a communication link. The set of nodes and communication links has the structure of a graph, called the communication graph, and they compose the network under study. In order to communicate, the nodes need to spend resources and the total amount of available resources is a local property of the nodes. Specifically, a node i may have to spend c_{ij} units of energy in order to transmit a message to node j. In this instance, assuming that the total energy available per node is limited, the nodes have to wisely use the available communications links in order to maximize the functional lifetime of the network. When a wireless sensor network monitors an area, the events that are detected near a sensor i must continuously be reported to the base station or sink, and this generates a fraction g_i of the total flow of information f. In other words, $g_i \cdot f$ messages per second are generated by sensor i and $\sum_{i=1}^{N} g_i = 1$. We denote f_{ij} the flow of messages from node i to node j with the convention that the sink is numbered 0, i.e. f_{i0} is the flow from node i to the sink. Finding the flow $\{f_{ij}\}$ which maximizes the lifetime of the sensor network amounts to solving the following problem:

Problem 1. Maximize the duration T such that $\forall i = 1, \ldots, N$

$$g_i \cdot f + \sum_{j=1}^{N} f_{ji} = \sum_{j=0}^{N} f_{ij} \quad \text{and} \tag{1}$$

$$T \sum_{j=0}^{N} f_{ij} c_{ij} \leq b_i \tag{2}$$

Equation (1) represents the constraints ensuring that $\{f_{ij}\}$ is a flow and Equation (2) ensures that in the duration T no sensor consumes more than its available energy b_i. If we proceed to the change of variables $\tilde{f_{ij}} = T f_{ij}$ ($\tilde{f_{ij}}$ is the total amount of messages sent from i to j in the duration T), and $\tilde{f} = Tf$, we get the following linear program:

Problem 2. Maximize the amount of reported events $\tilde{f}$ such that $\forall i = 1, \ldots, N$

$$g_i \cdot \tilde{f} + \sum_{j=1}^{N} \tilde{f_{ji}} = \sum_{j=0}^{N} \tilde{f_{ij}} \quad \text{and} \tag{3}$$

$$\sum_{j=0}^{N} \tilde{f_{ij}} c_{ij} \leq b_i \tag{4}$$

We emphasize the fact that Problem 1 considers the rate of data while Problem 2 considers the total amount of data collected by the sink. Another difference is that in 1 the total flow f is fixed while $\tilde{f}$ is a free variable. This distinction appears again when we discuss a weak optimal Pareto approach to the problem. We will now formally prove that maximizing T amounts to maximizing $\tilde{f}$.

Proposition 1. *Problem 1 is equivalent to Problem 2, which means that* $\{f_{ij}\}, T$ *is an optimal solution to the first if and only if* $\{\tilde{f_{ij}} = T f_{ij}\}, \tilde{f} = Tf$ *is an optimal solution to the second.*

Proof. We proceed by contradiction. Let us assume that $\{f_{ij}\}, T$ is an optimal solution to Problem 1 and that $\{\tilde{f_{ij}} = T f_{ij}\}, \tilde{f} = Tf$ is not an optimal solution to Problem 2. Then, there is solution $\{\tilde{g_{ij}}\}, \tilde{g}$ to Problem 2 such that $\tilde{g} > \tilde{f}$. We consider $T' > T$ such that $\tilde{g} = T'f > \tilde{f}$ and define $\{g_{ij}\} = \{\frac{1}{T'}\tilde{g_{ij}}\}$. We check directly that $\{g_{ij}\}$, T' is a feasible solution to Problem 1 with $T' > T$ which contradicts the optimality of T. We proceed similarly to show the other direction of the equivalence.

This proposition shows that maximizing the lifetime of the network is equivalent to maximizing the total number of messages gathered by the sink. Problem 2 corresponds to a max-flow problem where constraints are placed on the nodes. We now define what is an energy-balanced flow and propose a sufficient set of conditions ensuring that an energy-balanced flow maximizes the lifetime of the network.

Definition 1. *A flow $\{f_{ij}\}$ is called energy-balanced if there is a constant k such that for all $i = 1, \dots, N$ we have $\sum_{j=0}^{N} f_{ij}c_{ij} = kb_i$.*

Proposition 2. *If for all $i = 1, \dots, N$ there is λ_i such that*

$$\lambda_i \geq 0 \quad \textit{and}$$

$$-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} \geq 0,$$

and if there is an energy-balanced flow $\{f_{ij}\}$ such that

$$f_{ij} \cdot \left(-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0}\right) = 0 \tag{5}$$

then there is a duration T such that $\{f_{ij}\}, T$ is a solution to Problem 1.

Proof. We consider a path decomposition P of the flow $\{f_{ij}\}$, and a path $i_1, i_2, \dots, i_k$ of this decomposition from a sensor $i = i_1$ towards the sink $i_k = 0$, we can see from Equation (5) that

$$\lambda_{i_1} c_{i_1 0} = \lambda_{i_1} c_{i_1 i_2} + \lambda_{i_2} c_{i_2 i_3} + \ldots + \lambda_{i_{k-1}} c_{i_{k-1} i_k} \tag{6}$$

We want to compute the sum $\sum_i \lambda_i \sum_j f_{ij} c_{ij}$ by decomposing the flow into the paths followed by messages to the sink. For each such path p from i to the sink, Equation (6) shows that the contribution to the sum is $\lambda_i f_p c_{i0}$ where f_p is the number of messages per second flowing through the path p in the decomposition P. Since $g_i \cdot f$ is equal to the sum $\sum_{p \text{ from } i \text{ to } 0} f_p$, we get

$$\sum_i \lambda_i \sum_j f_{ij} c_{ij} = \sum_i \lambda_i g_i f c_{i0}.$$

On the other hand, any other solution $\{f'_{ij}\}, T'$ to Equations (1) and (2) uses paths satisfying the equation

$$\lambda_{i_1} c_{i_1 0} \leq \lambda_{i_1} c_{i_1 i_2} + \lambda_{i_2 i_3} c_{i_2 i_3} + \ldots + \lambda_{i_{k-1}} c_{i_{k-1} i_k}$$

which leads to

$$\sum_i \lambda_i \sum_j f'_{ij} c_{ij} \geq \sum_i \lambda_i g_i f c_{i0}$$

We conclude by using the assumption that the flow $\{f_{ij}\}$ is energy-balanced and by calling $T = \frac{1}{b_i}\sum_{j=0}^{N} f_{ij}c_{ij}$. Indeed,

$$f \cdot T \sum_i \lambda_i g_i c_{i0} = T \sum_i \lambda_i \sum_j f_{ij}c_{ij} = \sum_i \lambda_i b_i \geq T' \sum_i \lambda_i \sum_j f'_{ij}c_{ij} \geq f' \cdot T' \sum_i \lambda_i g_i c_{i0}$$

which shows that $T \geq T'$.

Actually, what we have shown is that if a communication graph contains only edges (i,j) such that $-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} = 0$, then any energy-balanced flow will maximize its lifetime. In other words, our result shows that edges (i,j) such that $-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} > 0$ are useless to increase the efficiency of the network. In a former work [8], appropriate λ_i values were computed in an ad-hoc way under the specific condition that the energy consumption grows quadratically with the range of emission.

3 Weak Pareto Optimality and Energy-Balanced Flows

In this section, we introduce weak Pareto optimality (Definition 2) and prove that for energy-balanced flows in sensor networks, weak Pareto optimality is equivalent to maximizing the lifetime of the network (Proposition 3).

Assuming a flow $\{f_{ij}\}$ of information towards the sink of a sensor network, the rate of energy consumption of a sensor i is given by $\sum_j f_{ij}c_{ij}$ and the lifetime of this sensor is inversely proportional to this rate, i.e. given by $b_i / \sum_j f_{ij}c_{ij}$. To maximize the lifetime of the sensors we have to solve the multiobjective optimization problem:

Problem 3. Minimize $\{i : \sum_j f_{ij}c_{ij}\}$ such that $\forall i = 1, \ldots, N$

$$f \cdot g_i + \sum_{j=1}^{N} f_{ji} = \sum_{j=0}^{N} f_{ij}$$

The weak Pareto approach, proposed in [3], is particularly well suited to study such multiobjective optimization problems.

Definition 2 (weak Pareto optimal flow). *A flow $\{f_{ij}\}$ is weak Pareto optimal if and only if there does not exist any flow $\{f'_{ij}\}$ such that $\sum_i f'_{i0} = \sum_i f_{i0}$ and*

$$\sum_{j=0}^{N} f'_{ij}c_{ij} < \sum_{j=0}^{N} f_{ij}c_{ij}, \quad \forall i = 1, \ldots, N \tag{7}$$

Intuitively this means that given a weak Pareto optimal solution it is not possible to increase the lifetime of the network since this would need to reduce the energy consumption of all sensors simultaneously. In [3], an algorithm is suggested to compute a flow such that the lifetime of all the sensors is the same and which produces a best approximation if no solution exists.

In the following, we prove that by looking only at energy-balanced flows, maximizing the lifetime of the network (Problem 1) is equivalent to finding a weak Pareto optimal solution to Problem 2. We emphasize that the equivalence is proved for energy-balanced propagation scheme.

Proposition 3. *An energy balanced flow (Definition 1) maximizes the lifetime of the network (Problem 1) if and only if it is a weak Pareto optimal solution (Definition 2) to the multiobjective optimization problem (Problem 3).*

Proof. We first prove that an energy balanced flow $\{f_{ij}\}$ that maximizes the lifetime T of the network is a weak Pareto optimal solution to (3). Assume that $\{f_{ij}\}$ is not weak Pareto optimal: then there is a flow $\{f'_{ij}\}$, such that $\sum_i f'_{i0} = \sum_i f_{i0}$ and

$$\sum_{j=0}^{N} f'_{ij} c_{ij} < \sum_{j=0}^{N} f_{ij} c_{ij}, \quad \forall i = 1, \ldots, N \tag{8}$$

The flow $\{f_{ij}\}$ satisfies the energy constraints (2), hence using (8) we have

$$T \sum_j f'_{ij} c_{ij} < b_i, i = 1, \ldots, N.$$

Therefore there is $T' > T$ such that $\{f'_{ij}\}, T'$ is a solution to Problem 1. This contradicts the optimality of the flow $\{f_{ij}\}$.

We next show that an energy-balanced flow which is a weak Pareto optimal solution solves Problem 1. We proceed by contradiction and assume that the flow $\{f_{ij}\}$ is energy-balanced and weak Pareto optimal but does not maximize the lifetime of the network. Then, there is a solution $\{f'_{ij}\}, T'$ to Problem 1 such that $T' > T$. The energy constraints satisfied are

$$T \sum_{j=0}^{N} f_{ij} c_{ij} = b_i, \quad \text{and} \quad T' \sum_{j=0}^{N} f'_{ij} c_{ij} \leq b_i, \quad \forall i = 1, \ldots, N.$$

Therefore, we have

$$\sum_{j=0}^{N} f'_{ij} c_{ij} \leq \frac{T}{T'} \sum_{j=0}^{N} f_{ij} c_{ij}$$

which shows that the flow $\{f_{ij}\}$ is not weak Pareto optimal.

4 Optimal Communication Graphs

We have seen that the good structures of the communication graphs on which optimal energy-balanced flows exist can be characterized (Proposition 2). In this section, we broaden this characterization by using the fact that optimal energy-balanced flows are also weak Pareto solutions to the multiobjective optimization problem (Problem 3).

Given a weak Pareto optimal solution of a multiobjective optimization problem, there is a way to formulate the problem as a classical linear program such that the optimal solution is the weak Pareto optimal solution [6,1]. Precisely, given a weak Pareto optimal solution to Problem 3 there is $\{\lambda_i\}$, such that $\sum_{i=1}^N \lambda_i = 1$, such that $\forall i = 1, \ldots, N$, $\lambda_i \geq 0$, and such that the weak Pareto solution coincides with the optimal solution of the following linear problem:

Problem 4. Minimize $\sum_{i=1}^N \lambda_i \sum_{j=0}^N f_{ij} c_{ij}$ such that $\forall i = 1, \ldots, N$

$$f \cdot g_i + \sum_{j=1}^N f_{ji} = \sum_{j=0}^N f_{ij} \tag{9}$$

$$f_{ij} \geq 0 \quad \forall j \tag{10}$$

We can readily prove that any solution to Problem 4 is a weak optimal solution to Problem 3 for any $\lambda_i \geq 0$. This linear problem is easier to study by making the $\{f_{ij}\}$ independent. We remove their dependency by replacing f_{i0} with $f \cdot g_i + \sum_j f_{ji} - \sum_j f_{ij} \geq 0$. We can then consider the following linear program:

Problem 5. Minimize $\sum_{i=1}^N \lambda_i \big[\big(f \cdot g_i + \sum_{j=1}^N f_{ji} - \sum_{j=1}^N f_{ij}\big) c_{i0} + \sum_{j=1}^N f_{ij} c_{ij}\big]$ such that

$$f \cdot g_i + \sum_{j=1}^N f_{ji} - \sum_{j=1}^N f_{ij} \geq 0, \ \ \forall i = 1, \ldots, N.$$

The dual of Problem 5 is

Problem 6. Maximize $\sum_{i=1}^N \big(\lambda_i g_i c_{i0} - \beta_i g_i\big)$ such that

$$-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} - \beta_j + \beta_i \geq 0, \ i, j = 1, \ldots, N.$$

By classical duality theory [9], we have the following equation:

$$\sum_i \lambda_i \big[\big(f \cdot g_i + \sum_j f_{ji} - \sum_j f_{ij}\big) c_{i0} + \sum_j f_{ij} c_{ij}\big] \geq f \sum_i (\lambda_i g_i c_{i0} - \beta_i g_i) \tag{11}$$

and the complementary slackness conditions are given by

$$f_{ij} \big(-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} - \beta_j + \beta_i\big) = 0, \ i = 1, \ldots, N. \tag{12}$$

$$\beta_i \big(f \cdot g_i + \sum_j f_{ji} - \sum_j f_{ij}\big) = 0, \ \ i = 1, \ldots, N \tag{13}$$

Definition 3. *A communication graph Λ is energy-optimal if there exists constants $\lambda_i \geq 0, \ i = 1, \ldots, N$ and $\beta_i \geq 0, \ i = 1, \ldots, N$ such that $-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} - \beta_j + \beta_i = 0$.*

A subgraph Λ of Γ is energy-optimal in Γ if Λ is an energy-optimal path and $-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} - \beta_j + \beta_i > 0$ for all edges $(i, j) \in \Gamma \setminus \Lambda$.

Precisely, the complementary slackness conditions (12) and (13) say that

– An energy-balance flow on the top of an energy-optimal communication graph Λ is maximal.
– If Λ is energy-balance optimal in Γ then using edges in $\Gamma \setminus \Lambda$ cannot improve the flow of data (optimal subgraph property).
– In any case, direct transmissions to the sink are possible and optimal only if $\beta_i = 0$.

Proposition 4. *The set of conditions defining an energy-balanced optimal communication graph is a convex set.*

Proof. Given $\{\lambda_i\}$, $\{\beta_i\}$ and $\{\lambda_i'\}$, $\{\beta_i'\}$ satisfying the equations

$$-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} - \beta_j + \beta_i = 0$$

and

$$-\lambda_i' c_{i0} + \lambda_i' c_{ij} + \lambda_j' c_{j0} - \beta_j' + \beta_i' = 0$$

it is readily checked that $\{\bar{\lambda}_i = p\lambda_i + (1-p)\lambda_i'\}$, $\{\bar{\beta}_i = p\beta_i + (1-p)\beta_i'\}$ with $0 < p < 1$ satisfies the equation

$$-\bar{\lambda}_i c_{i0} + \bar{\lambda}_i c_{ij} + \bar{\lambda}_j c_{j0} - \bar{\beta}_j + \bar{\beta}_i = 0.$$

Energy-balance optimal communication graphs are algebraically characterized and have the important property that energy-balance flows are also maximal. This property is necessary and sufficient, as stated in the following proposition.

Proposition 5. *An energy-balanced optimal flow determines an energy-balanced optimal communication graph.*

Proof. An energy-balanced optimal flow is equivalent to a Pareto optimal solution to Problem 3 or Problem 4 by Proposition 3. The complementary slackness conditions (12) and (13) are then satisfied (for suitable λ_i, β_i) and characterize an energy-balance optimal communication graph.

Proposition 6. *If the optimal flow satisfies $f_{i0} = f \cdot g_i + \sum_j f_{ji} - \sum_j f_{ij} > 0$, then $\beta_i = 0,\ i = 1, \ldots, N$.*

Proof. This is due to the slackness condition expressed in Equation (13).

Two remarks can be made to conclude this section. First, the conditions stated in Equation (12) restrict the set of non-vanishing f_{ij} in the optimal solutions. Then, the λ_i values depend on the particular Pareto optimal solution we search for.

5 Examples of Energy-Balance Optimal Communication Graphs

In this section, we present two examples of structures that allow for the existence of energy-balance optimal flows.

We first expose simple sufficient conditions ensuring the optimality of energy-balance flows. If we set $\beta_i = 0$ in Equations (12) and (13), the optimal solution to both Problems 4 and 6 is equal to $f \cdot \sum_i \lambda_i g_i c_{i0}$ (found by replacing f_{ij} with 0 in Problem 5) and the constraints set in Problem 6 read

$$-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} \geq 0, \; i = 1, \ldots, N. \tag{14}$$

The complementary slackness conditions set in Equation (12) now read

$$f_{ij} \left(-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{j0} \right) = 0, \tag{15}$$

Thus, f_{ij} can be non-vanishing only if $-\lambda_i c_{i0} + \lambda_i c_{ij} + \lambda_j c_{ij} = 0$. We summarize this in the following proposition.

Proposition 7. *Let us assume that there exist constants λ_i with $\sum_i \lambda_i = 1$ satisfying Equation (14) and a flow $\{f_{ij}\}$, f satisfying the complementary slackness conditions (15). Then, the flow is a weak Pareto optimal solution to Problem 3.*

Proof. Conditions set in Equation (15) imply that the value function of Problems 5 and 6 is equal. Given any set of λ_i values, the solution to Problem 5 is always a weak Pareto optimal solution to Problem 3.

5.1 A First Energy-Balance Optimal Topology

We now turn to the application of Proposition 7. We consider the complete graph on the set of vertices. Each edge is assigned a weight c_{ij} which corresponds to the energy cost of transmitting through that edge. We assume that we are able to compute λ_i constants such that Equation (14) is satisfied. If we can generate an energy-balanced flow $\{f_{ij}\}$ such that the conditions (15) are satisfied then by Proposition 3 this flow is maximal. Moreover, by the discussion of Section 2 this flow also maximizes the lifetime of the network.

This is used in an ad-hoc way in the papers [7,8] where it is assumed that sensors can transmit data to their nearest neighbour or directly to the sink. In these papers, the energy required to send data to neighbours at distance ≤ 1 is constant and the same for all the sensors. Moreover, it is assumed that the energy needed for transmissions otherwise grows like the square of the transmission distance. A directed acyclic graph is built to transmit data from any sensor to a unique sink by decomposing the network in slices corresponding to nodes distance from the sink to an equal number of hops, as shown in the left of Figure 1. The nodes belonging all to any particular slice are identified as a *super* node. This leads to the topology shown in the right of Figure 1. Normalizing the distances such that the transmission power to the nearest neighbours is one, explicit values for the λ_i constants are provided, i.e. $\lambda_i = \frac{1}{i(i+1)}$. Proposition 7 (and the results contained in the cited papers) shows that an energy-balanced flow using only one hop communications or direct transmissions to the sink maximizes the flow of data.

In fact, assuming that sensors use only two levels of energy in their transmissions, corresponding to short range transmissions to close neighbours and to

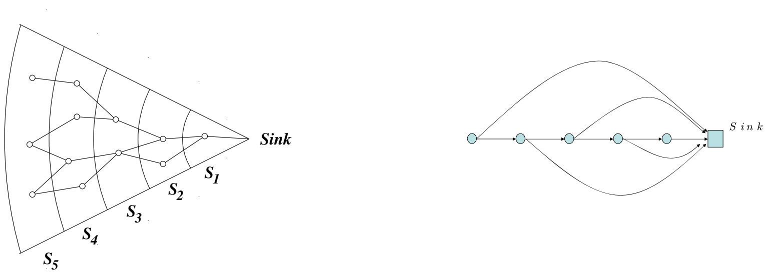


Fig. 1. An energy-balance optimal topology on the left, sensors transmit data to the next slice or directly to the sink (not represented). A simple model of it on the right.

long range transmissions to the sink, we prove that we can always compute λ_i satisfying the hypothesis on proposition 7.

Proposition 8. *We assume that the sensors composing the network transmit with two levels of energy, i.e. sensor i transmits with energy level c_i to a close neighbour or with c_{i0} to the sink with $c_i = c_j$ and $c_{i0} > c_i$, for all $i = 2, \ldots, N$ and $j = 1, \ldots, N$ (for sensors belonging to the first slice $c_i = c_{i0}$). Then, there exists λ_i satisfying the hypothesis of proposition 7. The transmission graph is directed to the sink and any sensors transmit either to the sink or to sensors belonging to the next slice (see the left side of Figure 1). The λ_i's values are equal for sensors belonging to the same slice.*

Proof. We first set arbitrarily $\lambda_i = 1$ for sensors belonging to the first slice. Using Equation (14) we recursively compute the values λ_j for sensors that are one more hop away from the sink. Because $c_{i0} > c_i$, we have $\lambda_i > 0$. At the end, we normalize to $\sum_{i=1}^{N} \lambda_i = 1$.

5.2 A Second Energy-Balance Optimal Topology

We now present another energy-balance optimal topology. The network is again divided into slices of width r. Sensors can transmit either to sensors belonging to the next slice or to sensors belonging to the next slice following the next slice (two slices away towards the sink). The energy costs are respectively c_1 and c_2, $c_2 > c_1$, independently of the slice the sensor belongs to. Sensor belonging to the second slice away from the sink transmit to the first slice with energy consumption c_1 or directly to the sink with energy c_2. Sensors belonging to the first slice have no other choice than to transmit to the sink with energy consumption c_1. Figure 2 illustrates this energy-balance optimal topology.

Proposition 9. *The topology described above and depicted on Figure 2 is energy-balance optimal if $c_{i0} \geq c_1 + c_{i-1,0}(c_2 - c_1)/c_1$*

Proof. The coefficients λ and β depend only on the slice number S_i; we use λ_i and β_i to denote the coefficients of sensors belonging to the $i-$th slice. We have

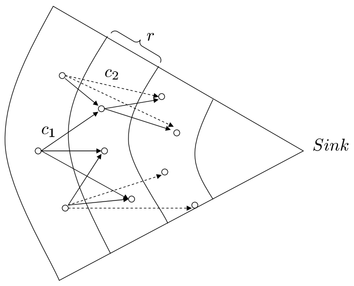


Fig. 2. An energy-balance optimal topology on the left, sensors transmit data to the next slice with energy consumption c_1 or two slices away with energy consumption c_2

to prove that there exist coefficients λ_i and β_i such that the conditions stated in Definition 3 are satisfied. To each sensor belonging to the first slice we assign a value $\lambda_1 \neq 0$ and $\beta_1 = 0$. The value λ_1 will be defined by normalizing the λ_i such that $\sum \lambda_1 = 1$. The value $\beta_1 = 0$ is necessary since sensors transmit data to the sink as stated by Proposition 6.

For sensors belonging to the second slice, we assign a value λ_2 solution to the equation $-\lambda_2 c_2 + \lambda_2 c_1 + \lambda_1 c_1 = 0$ and $\beta_2 = 0$. We have $\lambda_2 \geq 0$ since $c_2 > c_1$.

For sensors belonging to another slice we compute the parameters λ_i and β_i recursively. Let us assume that the coefficients are computed for all slices $1, 2, \ldots, i-1$. We determine λ_i and β_i as solution to the system of equations

$$-\lambda_i c_{i0} + \lambda_i c_1 + \lambda_{i-1} c_{(i-1)0} - \beta_{i-1} + \beta_i = 0 \tag{16}$$

$$-\lambda_i c_{i0} + \lambda_i c_2 + \lambda_{i-2} c_{(i-2)0} - \beta_{i-2} + \beta_i = 0 \tag{17}$$

By subtracting equation (16) to (17) we get

$$\lambda_i \underbrace{(c_2 - c_1)}_{>0} + \underbrace{\lambda_{i-2} c_{i-2,0} - \lambda_{i-1} c_{i-1,0} + \beta_{i-1} - \beta_{i-2} = 0}_{=-\lambda_{i-1} c_1 < 0},$$

where the second underbraced term is $-\lambda_{i-1} c_1$ by recurrence hypothesis (see Equation (16) where $i-1$ is substituted by $i-2$ and i by $i-1$) and leads to $\lambda_i > 0$. We then have $\lambda_i = c_1/(c_2 - c_1)\lambda_{i-1}$.

To see that $\beta_i \geq 0$ we use Equation (16) recursively. We know that $\beta_1 = 0$ for sensors in the first slice. We assume that $\beta_j \geq 0$ for $j = 1, \ldots, i-1$. Equation (16) states that $\beta_i \geq \lambda_i c_{i0} - \lambda_i c_1 - \lambda_{i-1} c_{i-1,0}$ which is positive by the assumption on c_{i0}.

6 On the Existence of Energy-Balanced Probabilistic Mechanisms

We proposed in the previous section reasonable network structures which are energy-balance optimal (Propositions 8 and 9). In this section, we discuss the existence of distributed energy-balance routing schemes on these topologies. We take advantage of the fact that energy-balance is a local property. Indeed, any node composing the network can check the balance of energy by comparing its energy consumption to that of its neighbouring nodes. We exploit this local property to construct an energy-balanced flow while routing data in a distributed manner.

6.1 A First Online Distributed Algorithm

We first consider the energy-balance optimal network structure defined in Proposition 8 with the additional constraint that the energy to transmit to the next slice is the same for any slice, i.e. in terms of Proposition 8 we have $c_i = c$, $\forall i$. We also assume that the communication channels are bidirectional and that while transmitting a message the nodes add information about their current level of energy consumption to it. This mechanism ensures that any node is aware of the level of energy consumption of its neighbouring nodes. Moreover, we assume that the initial level of available energy is the same for all the nodes.

Upon reception of a message, node i forwards the message to the neighbouring node j $(i \rightarrow j)$ with probability p_{ij} or directly to the sink with probability p_{ii}. The probabilities are computed online by using the following rule: *If the energy consumption of node j is larger than the average energy consumption of the neighbouring nodes then node i decreases p_{ij}, else p_{ij} is increased*[1], see Figure 3.

This algorithm is an application of stochastic approximation where we compute online the probability that the energy consumption of the neighbouring

```
variables:
i: the identifier of the current node
p_ij: the probability of transmitting to node j, p_ii the probability of transmitting to the sink.
x_i: the level of energy consumption of the current node.
x_j: the level of energy consumption of a neighbouring node j.
t_i: counts the number of messages sent by the current node
Initialise $p_{ij} = 1/deg_i$, where $deg_i$ is the degree of the current node and includes the node $i$ itself.
upon reception of a message
    $p_{ij} \leftarrow p_{ij} + \frac{1}{t_i}(\frac{1}{deg_i}\sum_{i\rightarrow k} x_k - x_j)$
    Normalize the $p_{ij}$ so that $0 \leq p_{ij} \leq 1$ and $\sum_j p_{ij} = 1$
    $t_i \leftarrow t_i + 1$
    select a node $j$ such that $i \rightarrow j$ with probability $p_{ij}$
    forward the message to the selected node or transmit directly to the sink if the selected node is $i$
end upon
```

Fig. 3. Pseudo-code of the program executed by the nodes with the topology define by Proposition 8

[1] This is implemented by computing $p_{ij} \leftarrow p_{ij} + \frac{1}{t_i}(\frac{1}{deg_i}\sum_{i\rightarrow k} x_k - x_j)$.

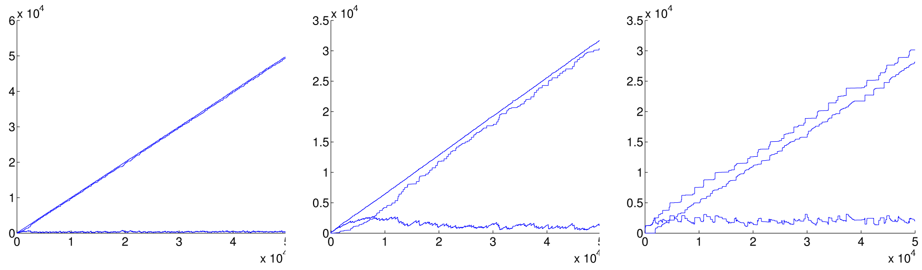


Fig. 4. Min-max energy consumption plots. From the left to the right the number of nodes is 30, 80, 130 (the maximal distance from a node to the sink is 1), the communication slice size are 0.3, 0.2, 0.1 and the nodes are scattered randomly.

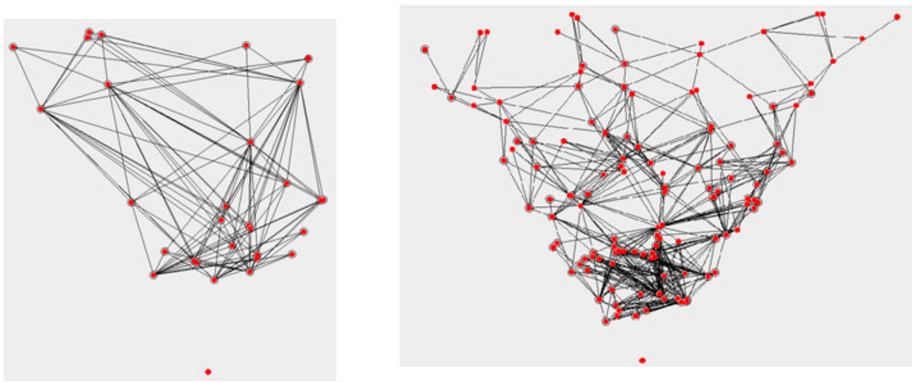


Fig. 5. The communication graphs of the two networks on the top of which were conducted the experiments. Only edges conveying more than 5% of the total traffic going out from a node are represented. On the left there are 80 nodes with size slice of 0.3 and on the left 130 nodes with size slice 0.1 (the maximal distance from the sink to a node is 1).

nodes is higher than the average. As such, it can be straightforwardly adapted to more general situations. A formal analysis of the properties of the algorithm might be done in this framework and is left for further work. Background material can be found in [2,13,10].

The numerical validation of the algorithm is presented on Figure 4. Messages are generated successively and once routed to the sink we record the maximal and the minimal levels of energy consumption among all the nodes. The plots show clearly that the energy growth is linear and that the difference between the maximal and the minimal energy consumption is bounded. The difference is also plotted. The plots represent the routing of 50′000 messages.

The particular communication graphs on the top of which the protocol was applied and produce the numerical results discussed above are plotted on Figure 5.

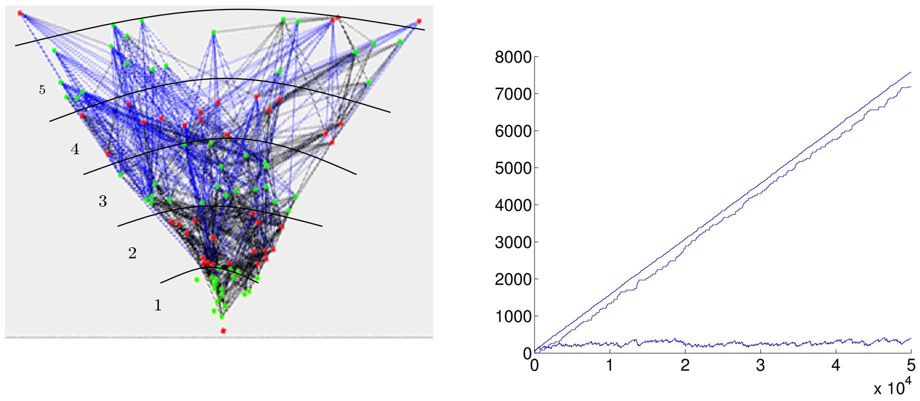


Fig. 6. Min-max energy consumption plots. From the left to the right the number of nodes is $30, 80, 130$ (the maximal distance from a node to the sink is 1), the communication slice size are $0.3, 0.2, 0.1$ and the nodes are scattered randomly.

6.2 A Second Online Distributed Algorithm

The second energy-balance optimal topology that we consider is the one defined by Proposition 9. The strategy to balance the energy consumption is the following: *If the current level of energy consumption of a node is larger than the average energy of its neighbours then it decreases its energy consumption by increasing the number of messages sent to the next slice (cost c_1) and decreasing the number of messages sent two slices away (cost $c_2 > c_1$). In the other case it proceeds by increasing the two slice away transmission (cost c_2) and decrease the transmission to the next slice (cost c_1).*

This mechanism is similar to the mechanism applied in the first proposed algorithm. However, nodes now have to balance the energy among nodes belonging to a same slice. For this purpose, each node computes the average energy consumption of the neighbouring nodes belonging to the next slice, $mean_1$, and two slices away $mean_2$. The inter-slice energy consumption is balanced with a similar strategy, i.e. each node transmits more messages to nodes whose energy consumption is smaller than the slice average ($mean_1$ or $mean_2$ depending on the position of the receiving node). More precisely, the probability that node i forward a message to node j is updated in the following way if we assume that node j belongs to the first slice

$$p_{ij} \leftarrow p_{ij} + \frac{1}{t_i}\big(\underbrace{(mean_1 - x_j)}_{\text{inter-slice}} + \underbrace{(x_i - \frac{1}{deg_i}\sum_{i\to k} x_k)}_{neighbour-balance}\big).$$

The first underbraced term balances the energy consumption of node j with respect to the others nodes belonging to the same slice. This is done by increasing/decreasing the probability of transmission to j if its energy consumption is

```
variables:
$i$: the identifier of the current node
$p_{ij}$: the probability of transmitting to node $j$.
$x_i$: the level of energy consumption of the current node.
$x_j$: the level of energy consumption of a neighbouring node $j$.
$t_i$: the number of message sent by the current node
Initialise $p_{ij} = 1/deg_i$, $deg_i$ is the degree of the current node and includes the node $i$ itself.
upon reception of a message
    compute $mean_1 = \frac{1}{N_i^1} \sum_{i \to_1 j} x_j$, $mean_2 = \frac{1}{N_i^2} \sum_{i \to_2 j} x_j$
    if $j$ belongs to the next slice $(i \to_1 j)$
        $p_{ij} \leftarrow p_{ij} + \frac{1}{t_i}((mean_1 - x_j) + (x_i - \frac{1}{deg_i} \sum_{i \to k} x_k))$
    else
        $p_{ij} \leftarrow p_{ij} + \frac{1}{t_i}((mean_2 - x_j) - (x_i - \frac{1}{deg_i} \sum_{i \to k} x_k))$
    Normalize the $p_{ij}$ so that $0 \leq p_{ij} \leq 1$
    $t_i \leftarrow t_i + 1$
    select a node $j$ such that $i \to j$ with probability $p_{ij}$ and forward the message
end upon
```

Fig. 7. Pseudo-code of the program executed by the nodes with the topology defined by Proposition 9

below/above the mean ($mean_1$). The second underbraced term balances the energy consumption of node i with its neighbouring nodes by increasing/decreasing the transmission to nodes belonging to the first slice (like j) if the energy consumption of i is above/below the neighbour's average energy consumption. It is straighforward that increasing/decreasing the probability of transmission to nodes belonging to the first slice implies that we decrease/increase the probability of transmission to nodes belonging two slices away.

We point out that in this case it is no longer evident that a node can know at each time the current energy consumption of nodes which are two slices away without requiring extra transmissions. Indeed, in our scenario, the current level of energy consumption is attached to messages sent by adding an extra field. Thus, a node can update the current level of energy consumption of a two-slices away neighbouring node only when it transmits with maximal power. This corresponds to the case where the scheme is an asynchronous scheme as discussed in [2].

The second set of numerical validation considers a network composed of 100 nodes and sliced with slice size 0.2[2].Nodes transmit with two levels of energy consumption. Transmission to nodes belonging to the next slice costs $c_1 = 10$ and to two slices away costs $c_2 = 40$. Only nodes belonging to the last three slices (number $6, 5, 4$) generate messages, whereas the others nodes only act as relays. The quantity of generated messages is uniform among nodes of the last three slices.

The pseudo-code of the program executed by the nodes is represented on Figure 7. We denote $i \to_1 j$ and $i \to_2 k$ the transmissions such that node j belongs to the next slice from i and k two slices away. $N_i^1 = \sum_{i \to_1 j} 1$, $N_i^2 = \sum_{i \to_2} 1$

[2] The maximal distance between two nodes is 1.

are the corresponding total number of nodes. Notice that nodes belonging to the first slice have no choice and transmit directly to the sink with energy cost c_1 while the nodes belonging to the second slice use the program on Figure 3 since they can transmit to the sink directly.

The results of the numerical validation are plotted on Figure 6. On the left side, the communication graph is plotted and communications links which are not effectively used are not displayed. We observe that nodes belonging to the last slices favor long range transmission while nodes closer to the sink favor small range transmission. This is due to the fact that the nodes closer to the sink have a larger number of messages to handle than the nodes belonging to the last slices. In the right side of Figure 6 we observe that the energy consumption grows linearly with time and that at any time the difference between the maximal and minimal levels of energy consumed remains bounded. This numerically validates the stability of the algorithm under the given conditions.

References

1. Aubin, J.-P.: Applied Functional Analysis. Wiley, IEEE (2000)
2. Borkar, V.S.: Stochastic Approximation: A Dynamical Systems ViewPoint. Cambridge University Press, Cambridge (2008)
3. Dagher, J.C., Marcellin, M.W., Neifeld, M.A.: A theory for maximizing the lifetime of sensor networks. IEEE Transactions on Communications 55(2), 323–332 (2007)
4. Duan, C., Fan, H.: A distributed energy balance clustering protocol for heterogeneous wireless sensor networks (2007)
5. Efthymiou, C., Nikoletseas, S.E., Rolim, J.D.P.: Energy balanced data propagation in wireless sensor networks. In: IPDPS (2004)
6. Geoffrion, A.M.: Proper efficiency and the theory of vector maximization. Journal of Mathematical Analysis and Applications 22(3), 618–630 (1968)
7. Giridhar, A., Kumar, P.R.: Maximizing the functional lifetime of sensor networks. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, Los Angeles, California (2005)
8. Jarry, A., Leone, P., Powell, O., Rolim, J.: An optimal data propagation algorithm for maximizing the lifespan of sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 405–421. Springer, Heidelberg (2006)
9. Karlin, S.: Mathematical methods and theory in games, programming and economics. Dover Publications, Mineola (2003)
10. Kushner, H.J., Yin, G.G.: Stochastic Approximation and Recursive Algorithms and Applications. Springer, Heidelberg (2003)
11. Leone, P., Nikoletseas, S.E., Rolim, J.D.P.: An adaptive blind algorithm for energy balanced data propagation in wireless sensors networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 35–48. Springer, Heidelberg (2005)
12. Li, C., Ye, M., Chen, G., Wu, J.: An energy-efficient unequal clustering mechanism for wireless sensor networks. In: Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, November 2005, p. 8 (2005)
13. Meyn, S.: Control Techniques for Complex Networks. Cambridge University Press, Cambridge (2008)

14. Ok, C., Mitra, P., Lee, S., Kumara, S.: Maximum energy welfare routing in wireless sensor networks. In: Akyildiz, I.F., Sivakumar, R., Ekici, E., Oliveira, J.C.d., McNair, J. (eds.) NETWORKING 2007. LNCS, vol. 4479, pp. 203–214. Springer, Heidelberg (2007)
15. Powell, O., Leone, P., Rolim, J.: Energy optimal data propagation in sensor networks. Journal on Parallel and Distributed Computing 67(3), 302–317 (2007)
16. Singh, M., Prasanna, V.: Optimal energy-balanced algorithm for selection in a single hop sensor network (2003)
17. Singh, M., Prasanna, V.K.: Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network. In: Proceedings of the PerCom 2003, pp. 50–59 (2003)
18. Soro, S., Heinzelman, W.: Prolonging the lifetime of wireless sensor networks via unequal clustering. In: Proceedings of the 5th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (IEEE WMAN 2005) (April 2005)
19. Yu, Y., Prasanna, V.K.: Energy-balanced task allocation for collaborative processing in wireless sensor networks 10(1-2), 115–131 (2005)

Programming Sensor Networks with State-Centric Services

Andreas Lachenmann, Ulrich Müller, Robert Sugar,
Louis Latour, Matthias Neugebauer, and Alain Gefflaut

European Microsoft Innovation Center GmbH, Aachen, Germany
{andreasl,ulrichm,rsugar,llatour,
mattneug,alaingef}@microsoft.com

Abstract. This paper presents the uDSSP ("micro DSSP") programming model which simplifies the development of distributed sensor network applications that make use of complex in-network processing. Using uDSSP, an application is composed of state-centric services. These services interact by accessing the state of other services or by subscribing to changes of that state. uDSSP supports heterogeneous networks that consist of PCs, resource-rich sensor nodes, and resource-limited nodes with just a few kilobytes of RAM. The evaluation uses a non-trivial application to compare it to Abstract Regions and Tenet.

1 Introduction

In the future, sensor network applications will emerge that do more complex in-network processing than the simple aggregation (min, max, average, etc.) of most applications today. For example, in an elderly care application a body-worn sensor node determines the patient's activity level whereas nodes in the kitchen and in the dining room cooperate to infer the activities of daily living such as cooking and eating. Together with other nodes throughout the home they form the application that keeps track of the patient's activities. This information is useful to detect changes in the patient's behavior, which could be an indication of health problems.

Creating such distributed applications with complex in-network processing is still a difficult task. If such an application is – like most real-world applications today – directly developed on top of a sensor network operating system, the developer becomes easily distracted by low-level details such as sending packets and message formats. Furthermore, since application components are often tightly coupled with static references to specific nodes, the reusability of individual parts of the application is very limited. Therefore, in this paper, we present the uDSSP ("micro DSSP") programming model and middleware that addresses these problems. With its runtime system and a set of standard services, it relieves the developer of many recurring tasks such as dealing with communication or discovering nodes.

In our terminology, an *application* spans the whole network and provides the desired functionality to the user. It can include different classes of devices such as TelosB and Imote2 nodes as well as PCs. Such an application consists of services that are distributed over different nodes. The application is formed by combining services and letting them interact. A uDSSP *service*, which is of similar granularity as a web service, provides a part of the application's functionality like sampling data or inferring the activities of an elderly person. Such a service is state-centric, i.e., it is built around its state and exposes this state to other services. An example for the state of a service is the data sampled and the result of the computation on that data (e.g., the activity of an elderly person). A service communicates with other services by invoking operations such as subscribing to state changes or retrieving the state.

By storing the results of their computation in their state and by exposing this state, the services facilitate a loosely coupled, subscription-based form of interaction. Thus, a service does not have to broadcast its results to all nodes or include static references in the code where to send them. This subscription-based interaction reduces coupling between nodes and fosters software reuse. It fits many sensor network applications that are event-based and transmit data only when it changes.

This model has been inspired by the ideas of the Decentralized Software Services Protocol (DSSP) [1] but has been tailored to the special characteristics of sensor networks. DSSP is a SOAP-based protocol with some pre-defined operations for state-centric services. uDSSP implements a subset of the DSSP operations. Instead of using SOAP, however, it encodes its messages in a small, binary representation.

We have implemented uDSSP for several platforms, including the .NET Framework on PCs, the .NET Micro Framework (e.g., for Imote2 nodes), as well as Contiki [2] and Mantis [3] (e.g., for TelosB nodes). In both .NET implementations and in Contiki, uDSSP uses 6LoWPAN for a seamless integration in existing IPv6 networks. Since no implementation of 6LoWPAN is available for Mantis yet, these nodes are limited to a custom mesh protocol. This protocol is also supported by our uDSSP implementations for .NET. All platforms share the same concepts and a common (application-level) network format. Therefore, it is possible to create distributed applications that incorporate different classes of devices.

The contribution of this paper is a novel, state-centric programming model to create sensor network applications with complex in-network processing. This model simplifies the development of such applications by fostering the reuse of parts of the application, by providing easy-to-use mechanisms for communication between services, and by including common functionality (e.g., to discover services). As shown in the evaluation, efficient real-world applications can be implemented with uDSSP.

The rest of this paper is structured as follows. Section 2 presents related work. In Section 3 we give an overview of our overall system. Section 4 details the use of the programming model. In Section 5, we then present evaluation results, including case studies of two non-trivial applications. Section 6 concludes this paper.

2 Related Work

With its network data types, nesC [4], the programming language used for TinyOS, supports the developer in creating distributed applications that incorporate different types of devices. However, the interaction is not handled by the runtime system but requires a custom implementation by the application developer – often with static references to the data's destination. A more high-level, network-wide programming abstraction is offered by database approaches such as TinyDB [5]. However, such an approach is limited to simple queries. Li *et al.* [6] added support for events to

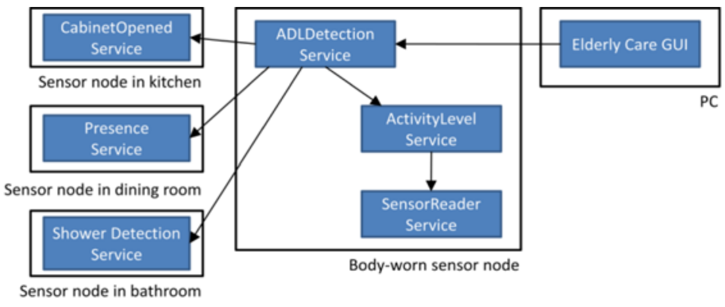


Fig. 1. Architecture of the elderly care application

DSware, another database-like middleware. However, they are limited to simple events that can be expressed in an SQL-like syntax. Macroprogramming languages such as Kairos [7] provide a way to create a network-wide program that is then executed in a distributed way in the sensor network. Unlike our approach, Kairos splits up a central program into individual threads and distributes them.

With its mechanism to register for certain data, TeenyLime [8] is similar to our subscription-based approach. Instead of exposing services as a structuring element, it shares data via a flat tuple space. Abstract Regions [9] uses a similar abstraction to exchange data. It forms groups of nodes based on properties such as hop distance or location. However, its data exchange mechanism is pull-based, i.e., nodes have to query the data of other nodes instead of being notified when it changes.

In Tenet [10], the sensor network consists of resource-limited motes and less constrained master nodes. This is similar to the network architecture of uDSSP. Unlike uDSSP, with Tenet, the application on the motes is written in a simple tasking language that is interpreted by the runtime system.

A design alternative for uDSSP would have been to use web services or Devices Profile for Web Services (DPWS) [11]. With its standard services, DPWS is similar to uDSSP but has not been designed for as resource-constrained devices as uDSSP: It requires the use of verbose XML messages. Tiny Web Services [12] have shown that web services can be implemented on resource-limited sensor nodes, though at the cost of some overhead. Instead of composing an application of services as with uDSSP, the architecture of Tiny Web Services seems to be targeted towards interaction with clients external to the sensor network.

3 System Overview

3.1 Architecture

Our network model is different from the early ideas of sensor network research with a flat, large-scale network of resource-limited nodes. Like most deployments today, the networks we target do not consist of thousands of nodes but rather of tens or hundreds. Furthermore, they can be heterogeneous and consist of different classes of devices. They include resource-constrained sensor nodes such as TelosB nodes, more

Table 1. Operations supported by uDSSP

Operation	Description
CREATE	Creates a new service instance
DROP	Removes the service instance
LOOKUP	Retrieves the partner services
GET	Retrieves the complete state
QUERY	Retrieves parts of the state
REPLACE	Replaces the complete state
UPDATE	Replaces parts of the state
SUBSCRIBE	Subscribes to state changes
SUBMIT	Calls an exposed function

Table 2. Example service definition

Item	Values
Service name	ShowerDetectionService
Service ID	0x45218A74
State	Bool Showering UInt16 HumidityLevel
Functions	-
Partner services	-

powerful nodes such as Imote2 nodes as well as PCs or servers. In this model, PCs are not just data sinks but fully participate in the network.

In our architecture, an application is composed of instances of services that communicate with each other. Each service instance has its state, a set of partner services, and functions that can be called by other services.

The overall application can be viewed as a graph of service instances, as shown in Fig. 1 for the elderly care application that will be used in Section 5.3.1. The arrows in the figure show the relationship to partner services, which tell a service which services it should, for example, subscribe to. The partners can be located on the same node or remotely in the network. In such an application, there is not necessarily a need for a single node to keep track of all service instances. Therefore, each service only has to have local knowledge about the services it interacts with.

The explicit definition of the service state distinguishes uDSSP services from web services. We think that exposing state fits well with the typical services we expect to find in a sensor network. For example, such services can make available their latest computation results on sensor data. Especially with our subscription mechanism, this model helps to compose the application from individual, loosely coupled services.

3.2 Runtime System

The main tasks of the runtime system are to handle requests, manage subscriptions, and deal with communication between services. There are two layers of the runtime system. The first one is service-specific and generated from the service definition. To access other services, this layer also includes generated proxies. We tried to keep this layer minimal to avoid duplicate functionality. The second layer is independent of the service. It dispatches messages to services and manages subscriptions.

In uDSSP, services invoke operations on other services. These operations are a subset of the operations specified by DSSP. Table 1 gives an overview of them. As described in Section 4, almost all operations are handled by the runtime system; the developer does not have to implement these standard tasks.

The CREATE and DROP operations are used to create a new instance of a service and to remove an existing one, respectively. The LOOKUP operation retrieves the set of partners of the service instance. GET, QUERY, REPLACE, UPDATE, and SUBSCRIBE all deal with the state of the service. These operations can be used to retrieve or modify either the complete state or parts of it. Furthermore, the SUBSCRIBE operation is used to subscribe for a notification when the state changes. This operation has a bitmask parameter, where each bit corresponds to a state variable. A bit set in the mask indicates that the client wants to be notified upon any modification of the corresponding subset of the state. Such a bitmask is also used to select variables with QUERY and UPDATE. Because of the limited capabilities of sensor nodes, we selected this simple but efficient model. The last operation is the SUBMIT operation. It is used to call a function that has been exposed by the service which is not directly related to the service's state. This mechanism is similar to remote procedure calls and web services.

3.3 Message Encoding

uDSSP messages are encoded in a binary format that is based on the service definition. Since we assume that two communicating services share knowledge about the service definition, the messages do not include any metadata but are just the concatenation of the data fields. In fact, the payload of some responses can only be decoded if the service specification and the original request are known. Compared to binary encodings of XML such as CBXML [13] this scheme reduces data size even more.

For instance, the size of an example QUERY message (excluding the headers of layers below) is just 14 bytes. In contrast, a SOAP-based DSSP message with similar functionality has more than 900 bytes even if each identifier consists just of a single byte (as suggested to reduce the size of web service requests [12]).

4 Using the Programming Model

In this section, we describe our programming model in more detail. First, we explain how a service is defined, what the compile-time tools generate from that definition, and how the service is implemented. Then we describe uDSSP's standard service and how an application is composed.

4.1 Defining Services

For our programming model, it is important to explicitly specify the state and interface of a service. The format of this description, however, is not a key component of our approach. In our implementation, a service is defined in a custom XML format.

Table 2 shows an example for the information contained in such a service definition. The name of the service ("ShowerDetectionService") is used to refer to the service in the source code. The ID, in contrast, is a (random) 32-bit value that is used to identify the service type in network messages. Both the name and the ID have to be unique in an application. While the ID identifies the service type and the corresponding message format (i.e., its interface), the service name is used to identify the implementation in the source code. At runtime, there can be several instances of a service running on a single node. Each instance is identified by the combination of the node's address, the service ID, and an automatically assigned instance number.

Besides simple data types such as integers of various lengths, uDSSP currently supports structs, arrays, and strings. In the example, the state consists of the "Showering" and "HumidityLevel" variables. No functions are exposed by this service.

```
public class ActivityLevelService : ActivityLevelServiceStub {
   /// Called when the service instance is created
  override protected void OnCreate(ServiceReference[] partners) {
     srProxy = new SensorReaderServiceProxy(this);
     srProxy.Bind(LOCAL_HOST, 0);
     srProxy.Notification += new SensorReaderServiceEventHandler
         (SensorData);
    SensorReaderServiceMask mask = new SensorReaderServiceMask();
    mask.HaveAccelX = true;
    mask.HaveAccelY = true;
    mask.HaveAccelZ = true;
    srProxy.Subscribe(mask, 0, 0, 100);
  }

  /// Notification called with new sensor data
  void SensorData(SensorReaderServiceProxy sender,
      SensorReaderServiceEventArgs e) {
    activityLevel = ComputeActivityLevel(e.State.accelX,
      e.State.accelY, e.State.accelZ);
    ActivityLevelServiceMask mask  = new ActivityLevelServiceMask();
    mask.HaveActivityLevel = true;
    NotifyUpdate(mask);
  }
  ...
}
```

Fig. 2. Sample of a .NET service implementation

4.2 Code Generation

A compile-time tool generates source code (C or C#) from the service definition. First, it generates the service-specific part of the runtime system. This code is responsible for message encoding and decoding as well as for handling all requests. If possible, it replies to the request without requiring interaction of user-created code.

Second, the code generator creates a client proxy for the service. Another service can bind this proxy to a specific instance and invoke operations on it. As the counterpart of the service-specific part of the runtime system, the proxy deals with communication. The user of the service only has to call a simple function of the proxy.

Finally, the code generator creates a template of the service implementation. The developer just has to fill the function prototypes given there. Those include the functions exposed in the service definition as well as callbacks that are invoked by the runtime system when some operations are performed. For example, the service can react to the case when no subscriber is present any longer and stop sampling data. If these callbacks are not needed, they can be left empty.

4.3 Implementing a Service

Reacting to events from the runtime system, adding the functionality of the service, and updating the state are the only parts the developer has to write. Common tasks, such as communicating with other services are handled by the runtime system.

Fig. 2 shows parts of a service implementation for the .NET Micro Framework. The implementations for Mantis and Contiki vary slightly because they are not object-oriented. The service is the ActivityLevelService, which determines the activity of an elderly person based on accelerometer data. The “OnCreate” function is called when instantiating the service. As specified with the “mask” variable, it subscribes to the three axes of acceleration data of the SensorReaderService on the same host.

“SensorData” is the function registered for receiving the notifications of the SensorReaderService. Here the service computes the activity level and notifies its subscribers of the state change. The “mask” parameter tells the subscribers which part of the state has been modified. Rather than sending automatic notifications after each change, the developer has to call the “NotifyUpdate” function in order to make sure that subscribers are only notified when the data is in a consistent state. The user code just modifies local variables and calls functions of the runtime system and proxies.

To simplify application development, all calls to other services are blocking. Therefore, the developer does not have to deal with replies that arrive asynchronously. The matching to the request is done by the runtime system. If no reply arrives, the blocking functions return with an error after a timeout. Since .NET and Mantis support multithreading, the implementation of synchronous communication is straightforward there. For Contiki, each such call is a so-called protothread [14].

4.4 Composing an Application

An application is composed by combining instances of services running throughout the network. As shown in Fig. 2, a connection between two services is created by instantiating a proxy and binding it to a service instance. The example refers to a static node address (in this case the local node). However, using the Discovery and Metadata Services (see Section 4.5), it is also possibly to find such partner services at runtime. Then it can, e.g., subscribe to all temperature services in the living room.

Other than binding to an existing instance, a service can use the proxy to create a new service instance – if the code for executing it is installed. In that case, it can pass references to other services as parameters. This way a generic service can be reused unmodified without the need for adding references to specific nodes in the code.

4.5 Standard Services

A key component of uDSSP is that it includes a set of standard services that we expect to be useful for many applications. We identified the following services by building several non-trivial applications:

Discovery Service: The Discovery Service detects nodes that have joined or left the network. Subscribers of this service can react to changes in the network and query more information from new nodes using the Directory Service and the Metadata Service. The Discovery Service is fully implemented on nodes with sufficient resources. On resource-constrained Mantis and Contiki nodes, it refers to another node.

Directory Service: The Directory Service keeps track of all service instances running on the local node. It returns their service IDs and instance numbers. More information can be retrieved by sending a LOOKUP request to the service instance.

Metadata Service: The Metadata Service links a node to the physical environment it is deployed in. It provides information about the sensor node’s identity, capabilities and location. The user can set the metadata upon deployment.

Table 3. Memory size of the runtime (in bytes)

	Program memory	RAM
Mantis	10,180	2,188
Contiki	13,055	1,108
.NET MF	44,808	5,346

Deployment Service: The Deployment Service is used for installing new services on a node at runtime (assemblies for .NET or dynamically linked ELF files for Contiki and Mantis). Since each node potentially executes different services, uDSSP cannot leverage existing code distribution mechanisms.

5 Evaluation

In this section, we present evaluation results from experiments with real sensor nodes. We present results about the memory footprint of uDSSP and its performance. Furthermore, we describe how uDSSP can be used in real-world applications and how it compares to other approaches.

5.1 Memory Footprint

Table 3 shows an overview of the memory consumption of uDSSP. These numbers are just the size of the runtime itself; they exclude the operating system, the network stack, and services. The Mantis and Contiki implementations consume just 10-13 KB of program memory and 1.1-2.2 KB of RAM. The Mantis implementation consumes more RAM since this number already includes the reserved stack space for the uDSSP threads. In Contiki, our implementation is based on stack-less protothreads [14]. Instead, since protothreads cannot use local variables, some (reentrant) functions use pointers to store their state in variables passed as parameters. This and the use of IPv6 with its longer addresses increase the code size of the Contiki implementation. For the .NET Micro Framework implementation size limitations are less stringent because it is executed on less constrained devices. Even there, uDSSP also needs just a few KB.

If services are added, the service-specific part of the runtime system consumes at least 2-3 KB of program memory. However, the actual size and also the size in RAM largely depend on the service itself. For example, the CabinetOpenedService, which will be described in Section 5.3.1, needs 2.6 KB of program memory on Mantis. If not the full functionality of uDSSP is needed, both the size of the runtime system and the size of the generated code can be reduced further by deactivating optional functionality with *ifdefs*.

5.2 Performance of the Runtime System

Although we expect that services typically do not send requests at a high rate, the time for processing requests gives a good indication about the performance of uDSSP's runtime system. To get these numbers, we repeatedly executed a GET request for arrays of different sizes. Although the results shown here are limited to

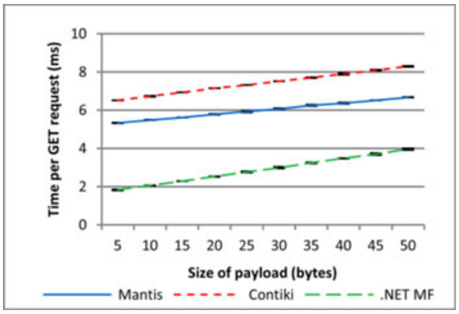


Fig. 3. Time in milliseconds for processing a local GET request, including (almost invisibly small) 95% confidence intervals

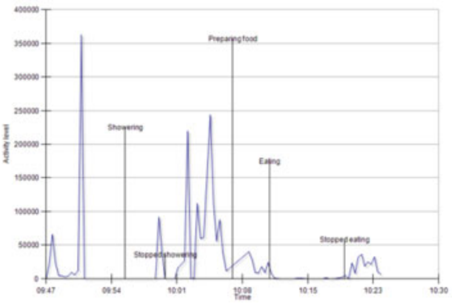


Fig. 4. Activities detected by the elderly care application

GET requests, the performance of the other request types supported by uDSSP are similar. For example, a local QUERY request takes about 1% longer than the GET request since the bitmask has to be included in the request and evaluated by the receiver.

Fig. 3 shows the results for requests within a single node. The Mantis and Contiki services were run on a TelosB node whereas the .NET Micro Framework implementation was executed on an Imote2. With increasing sizes of the state, the delay grows slightly by approximately two milliseconds. We attribute the difference between Mantis and Contiki to the use of IPv6 addresses in the Contiki implementation. In a real application that includes some other functionality, the difference would be smaller.

For remote requests, the processing time largely depends on the bandwidth of the radio channel and the MAC layer protocol. Since uDSSP is independent of those low-level communication mechanisms, these results are not meaningful to evaluate the performance of our runtime system. To give a general idea of the performance, depending on the platform and the size of the request, the processing time for a request to a node in the local neighborhood was measured between 29 and 43 ms.

Using Contiki's online energy estimation mechanism [15], we measured the energy overhead of an application on TelosB that sends a notification to a neighboring node every minute. Compared to a highly optimized, non-uDSSP application that implements the same functionality with a simple best-effort network protocol, the overhead of uDSSP is less than 0.07 mW, which should be negligible for most applications.

5.3 Case Studies

In this section we present two applications we have implemented with uDSSP. We use the first one to compare uDSSP with other programming models and the second application to show that it can be used to implement a wide variety of real-world applications. Both scenarios have been taken from the WASP project [16].

5.3.1 Elderly Care

With an aging population, improving care in the home of elderly people – instead of having them move to a nursing home – becomes more and more important. By monitoring their activities of daily living, sensor networks can help elderly people living alone and give their relatives the comforting information that they are doing well.

Fig. 1 in Section 3.1 shows the general architecture of the corresponding uDSSP application. The patient has a body-worn sensor node with an accelerometer to monitor the overall activity (e.g., attached to a wireless emergency button around the neck). We have selected an Imote2 for this part because it has more processing resources. Besides the body-worn node, the application consists of additional sensors nodes that are deployed throughout the home (e.g., TelosB nodes). We assume sensor nodes to be deployed in the kitchen, the dining room, and the bathroom. Using the input from these sensors, our application can detect if the patient is preparing a meal, eating, or showering. With the modular approach of uDSSP, this application could be easily extended with additional services.

The CabinetOpenedService sends a notification when the fridge or a cabinet in the kitchen has been opened. Our implementation of this service simply uses the light sensor which is available on most sensor nodes today. The PresenceService uses a pressure-sensitive foil on a chair to detect if somebody is sitting there. The ShowerDetectionService is deployed on nodes in the bathroom. If the humidity level is above a threshold, it assumes that somebody is having a shower. All of these services just notify their subscribers of state changes and do not send their raw data. Unlike scientific monitoring applications such as habitat monitoring, the users of this application are not interested in the raw sensor data. Therefore, it is sufficient if the services send messages when they detect an event and can truly benefit from in-network processing.

The application makes use of uDSSP's discovery functionality to incorporate several instances of each of these services. Using the standard services such as the Directory Service and the Metadata Service (see Section 4.5), it discovers services (e.g., the CabinetOpenedService) on all nodes and determines in which room the nodes have been deployed. Only if the location fits the expected room (i.e., in this case the kitchen), it subscribe to this service.

On the body-worn node, the SensorReaderService provides the data from the sensor board – in our case the acceleration values only – to the subscribers. The ActivityLevelService samples data at a frequency of 10 Hz, adds the values for the three axes of each sample together, and periodically computes the variance of the values. The results give a good indication of the patient's level of activity [17]. Finally, the ADLDetectionService uses the activity values and information from sensor nodes in the apartment to determine the patient's activities of daily living. For this purpose, it implements some simple rules such as if the fridge and the cabinets in the kitchen have been opened, the patient probably prepares a meal. The GUI subscribes to services running on the sensor node worn by the patient and displays its results.

We deployed this application in an apartment and monitored the activities of the person living there. Fig. 4 shows an example of the morning activities detected by the application in this deployment. Using the sensor nodes deployed throughout the house, the application detects the activities of showering, preparing a meal, and eating. Furthermore, it shows the activity level of the person with the body-worn sensor node. For example, the activity level also shows the reduced movements when the person sits at the table to eat breakfast. Even with the simple sensors we use, the application can give useful hints about the patient's status and activities.

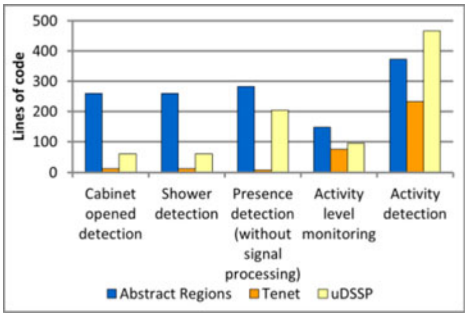


Fig. 5. Lines of code for the elderly care application

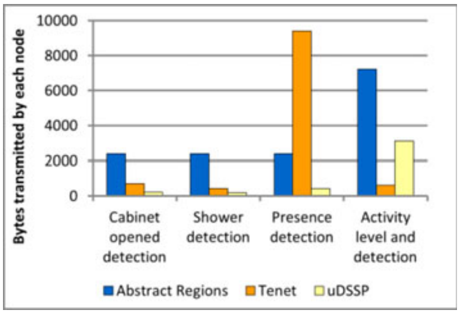


Fig. 6. Bytes transmitted in the elderly care scenario

To compare uDSSP with other approaches, we have implemented (almost) equivalent applications in other programming models: Abstract Regions [9] and Tenet 2.0 [10]. We tried to optimize all versions as much as possible regarding lines of code and bytes transmitted. The Abstract Regions version does not include support for multi-hop routing and discovery of other services; each node announces itself only in its radio range. In the Tenet implementation, cabinet and shower monitoring, which just check the threshold of a periodic sensor reading, can be easily implemented in Tenet's tasking language. However, these small programs are already very close to the maximum program size supported by Tenet. The complexity of presence detection, which requires some processing of the sensor signal, was too high to implement in this language. Therefore, the raw data has to be transmitted to the less constrained master node for processing in a C application. Finally, since the activity detection requires input from other nodes, this functionality is also not supported by Tenet's tasking language and has to be implemented on the master node in C. Unlike the size limitations, which could probably be modified, it is a fundamental principle of Tenet that only the less-constrained master nodes can process input from other nodes.

In Fig. 5 we compare the lines of user-written code to implement the application. For Abstract Region, these numbers do not include changes that were necessary to its runtime components in order to support nodes that are part of several regions. For Tenet, the numbers are quite low on the sensor nodes because the pure functionality of some nodes is readily available in its tasking language. However, as described above, some functions cannot be implemented in this language and have to be run on the master node. Furthermore, since Tenet's language requires the use of numbered attributes instead of meaningful variables, writing applications is more error-prone than the lines of code suggest. uDSSP is somewhere between Abstract Regions and Tenet. However, compared to Tenet, it provides access to the full functionality of a node and includes additional support for discovering new nodes. This discovery mechanism is mostly responsible for uDSSP's higher numbers for activity detection.

Fig. 6 compares the number of bytes sent in the first 40 minutes in an exemplary run of the sequence shown in Fig. 4. These numbers only include the bytes sent on the application layer because the underlying protocols are not important for this comparison. For Abstract Regions and uDSSP, the activity node transmits many bytes since the activity level is sent frequently to the GUI. These numbers are comparatively

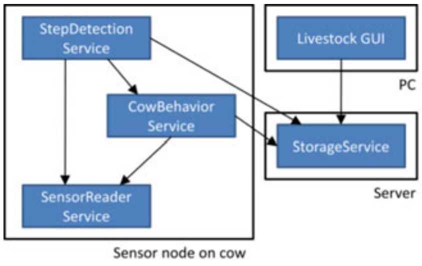


Fig. 7. Architecture of the livestock application

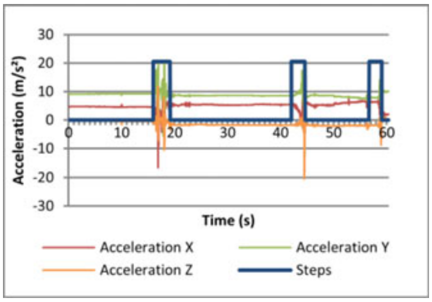


Fig. 8. Acceleration readings and steps detected

small for Tenet because they include only the pure application data in the messages sent by the C application. Unlike with uDSSP, the GUI application cannot benefit from the programming model and has to parse the messages manually. On the presence detection node, the numbers are significantly bigger for Tenet because the node has to transmit its raw data for processing to a master node.

Abstract Regions has significant overhead for two reasons. First, although we increased the interval from 1 s to 30 s, it still sends periodic beacons to announce itself to the other nodes in the neighborhood. Second, due to its pull-based data sharing approach, a node interested in data has to periodically query the data sources. In an application where data like the result of the shower detection changes very infrequently, the subscription-based model of uDSSP is preferable.

For uDSSP, these numbers include the overhead for discovering services and for subscribing to them (about 150 bytes for the cabinet node). In a static network, this is a one-time overhead. Without that, the shower detection node, for example, just sends 34 bytes. This number is comparable to an optimized manual implementation.

We are convinced that these results can be transferred to other applications. Tenet is suited well if the resource-limited nodes do only very simple processing and if the developer switches to its new, unfamiliar programming language. The pull-based model of Abstract Regions is of advantage if the observed data changes faster than updates are needed by the nodes interested in this data. uDSSP supports this use case also well by setting a minimum interval between notifications. Furthermore, it offers a good compromise between expressiveness of the programming language, complexity of the source code, and high efficiency for infrequent events.

5.3.2 Livestock Monitoring

Dairy farmers have to deal with claw health problems of their cows. If these problems are detected early, they can be treated before the cow is seriously impaired. However, to reduce costs, many farmers have to increase the size of their herds. Therefore, they have less time to spend with each cow. By continuously monitoring the activities of the cows with a wireless sensor network, less monitoring by the farmers is needed.

In this application, we focus on two aspects: the proportion of the time the cows are standing or lying and the number of steps they take. For this purpose, we attach a sensor node with an accelerometer to a leg of each cow. Using the accelerometer as a tilt sensor, we distinguish between cows that are standing and lying. Furthermore, by computing the variance of acceleration readings, we detect the steps of a cow.

Fig. 7 shows an overview of the services running in this application. There can be many sensor nodes present that run such services for one cow. The SensorReaderService interfaces with the node's sensor board. Two services subscribe to the acceleration data provided by this service: the CowBehaviorService, which monitors if the cow stands or lies, and the StepDetectionService, which determines the number of steps of the cow. These two services subscribe to the SensorReaderService at different notification rates (1 Hz and 50 Hz, respectively). When the cow is not standing, no steps have to be detected and the subscription of the StepDetectionService can be temporarily released. The SensorReaderService is notified of the currently needed maximum rate and can adjust its sampling rate accordingly. Both processing services deliver their results to a generic StorageService outside the sensor network. A GUI application retrieves the data from the storage service and subscribes to it in order to be notified when new data from a cow arrives. Alternatively, the GUI or an application-specific storage service could directly subscribe to the cow services.

To show the practicality of this application, we performed some experiments on a farm. Since there is always the risk of injuring the animal when attaching or removing the sensor node, in this application it is important that software updates can be installed wirelessly. With uDSSP, this is the task of the Deployment Service (see Section 4.5). Fig. 8 presents some results of the StepDetectionService and the corresponding raw acceleration data. In this example, there were three steps detected during one minute. It should be noted that the raw data shown here has been made available for testing. To reduce network traffic and energy consumption, the application only provides access to the behavior and number of steps detected.

6 Conclusions and Future Work

As we have described in this paper, uDSSP provides a programming model and the corresponding middleware to create applications that do complex in-network processing like the applications in Section 5.3. We are convinced that such applications will be among the first sensor network applications to be widely used. Using the abstraction of state-centric services, parts of the application can be developed independently and later be combined. Services do not have to deal with the users of their data. Other services that are interested in the data will be notified automatically or can retrieve it when needed. With its higher level of abstraction, uDSSP helps the programmer to focus on the actual functionality of the application.

The runtime system of uDSSP is small and efficient. Furthermore, the evaluation shows that it is possible to develop a wide range of applications with uDSSP. Compared to other approaches, uDSSP offers a good compromise of flexibility and efficiency.

With the same programming interface and communication protocols available on PCs, sensor nodes are no longer simple data suppliers attached to a serial port: They can invoke services outside the sensor network and fully participate in a network consisting of sensor nodes and IPv6-capable computers. Therefore, uDSSP will enable exciting new sensor network applications that integrate with other networks.

Regarding future work, we are planning to add security mechanisms that restrict the access to nodes and services. Especially if sensitive data like in the elderly care application is transmitted, this is necessary for the system to be accepted by the users.

Acknowledgments. This work is partially financed by the European Commission under the Framework 6 IST Project "Wirelessly Accessible Sensor Populations (WASP)".

References

1. Nielsen, H.F., Chrysanthakopoulos, G.: Decentralized Software Services Protocol – DSSP/1.0., http://purl.org/msrs/dssp.pdf
2. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: Proc. of the Worksh. on Embedded Netw. Sensors (2004)
3. Bhatti, S., et al.: MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. Mobile Networks and Applications 10, 563–579 (2005)
4. Gay, D., et al.: The nesC language: A holistic approach to networked embedded systems. In: Proc. of the Conf. on Programming Lang. Design and Impl., pp. 1–11 (2003)
5. Madden, S.R., et al.: TinyDB: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30 (2005)
6. Li, S., et al.: Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. Telecommunication Systems 26(2-4), 351–368 (2004)
7. Gummadi, R., Gnawali, O., Govindan, R.: Macro-programming Wireless Sensor Networks using Kairos. In: Proc. of the Conf. on Distrib. Comp. in Sensor Syst. (2005)
8. Costa, P., et al.: Programming Wireless Sensor Networks with the TeenyLime Middleware. In: Cerqueira, R., Campbell, R.H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 429–449. Springer, Heidelberg (2007)
9. Welsh, M., Mainland, G.: Programming Sensor Networks Using Abstract Regions. In: Proc. of the 1st Symp. on Networked Systems Design and Implementation, pp. 29–42 (2004)
10. Gnawali, O., et al.: The Tenet Architecture for Tiered Sensor Networks. In: Proc. of the Conf. on Emb. Netw. Sensor Syst., pp. 153–166 (2006)
11. Chan, S., et al.: Devices Profile for Web Services (2006)
12. Priyantha, N.B., et al.: Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In: Proc. of the Conf. on Emb. Netw. Sensor Syst., pp. 253–266 (2008)
13. Conner, M.: CBXML: Experience with Binary XML. In: W3C Workshop on Binary Interchange of XML Information Item Sets (2003)
14. Dunkels, A., et al.: Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. Proc. of the Int'l Conf. on Emb. Netw. Sensor Syst., 29–42 (2006)
15. Dunkels, A., et al.: Software-based On-line Energy Estimation for Sensor Nodes. In: Proc. of the Worksh. on Embedded Networked Sensors (2007)
16. WASP consortium: WASP project web site, http://www.wasp-project.org/
17. Lo, B., et al.: Real-Time Pervasive Monitoring for Postoperative Care. In: Proc. of the Worksh. on Wearable and Implantable Body Sensor Networks, pp. 122–127 (2007)

Fast Decentralized Averaging via Multi-scale Gossip

Konstantinos I. Tsianos and Michael G. Rabbat

McGill University, Department of Electrical and Computer Engineering
Montreal, QC, Canada
konstantinos.tsianos@gmail.com,
michael.rabbat@mcgill.ca

Abstract. We are interested in the problem of computing the average consensus in a distributed fashion on random geometric graphs. We describe a new algorithm called Multi-scale Gossip which employs a hierarchical decomposition of the graph to partition the computation into tractable sub-problems. Using only pairwise messages of fixed size that travel at most $O(n^{\frac{1}{3}})$ hops, our algorithm is robust and has communication cost of $O(n \log\log n \log \epsilon^{-1})$ transmissions, which is order-optimal up to the logarithmic factor in n. Simulated experiments verify the good expected performance on graphs of many thousands of nodes.

1 Introduction

Applications in sensor networks often demand that nodes cooperatively accomplish a task without centralized coordination. Autonomy is often equated with robustness and scalability in large-scale networked systems. This is especially true in wireless networks, where fundamental limits on spatial and temporal channel reuse limit the amount of communication possible at any instant in time. Moreover, when nodes are battery-powered—a typical design element of wireless sensor networks—each transmission consumes valuable energy resources. This has stimulated research into resource-efficient algorithms for distributed computing and coordination. Gossip algorithms are an attractive paradigm for decentralized, autonomous computation. A classic and well-studied example is gossiping to compute the average consensus: in a graph $G = (V, E)$ with $|V| = n$ nodes, where each node initially has a value $x_i(0)$, the goal is to compute an estimate of the average $x_{\text{ave}} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ at all nodes. At each gossip iteration, a random connected subset $S(t) \subset V$ of nodes exchange their current estimates, $x_i(t)$, and locally compute the update $x_i(t+1) = \frac{1}{|S(t)|}\sum_{j \in S(t)} x_j(t)$. In the original algorithm, described in [1] and revisited in [2] and [3], $|S(t)| = 2$ at every iteration, and the pair of nodes that update are neighbors in G.

Gossip algorithms have the attractive properties that they run asynchronously, do not require any specialized routing protocols, and therefore do not create bottlenecks or single points of failure. In the original gossip algorithm, pairs of nodes that communicate directly exchange information at each iteration. Consequently, no special routing is needed, and the algorithm has been shown to be robust to fluctuating availability of links, as well as other changes in topology.

Of course, robustness and autonomy come at a price. Standard pair-wise randomized gossip is inefficient on topologies commonly used to model connectivity in wireless networks, such as grids and random geometric graphs; the number of messages per node scales linearly with the size of the network. In contrast, one can imagine numerous other approaches to computing a linear combination of the values at each node, assuming the existence of some specialized routing such as a Hamilton cycle or spanning tree, in which the number of messages per node remains constant as the network size tends to infinity. Although the overhead involved in finding and maintaining these routes may be prohibitive, requiring more centralized control and creating bottlenecks and single points of failure, these observations have motivated substantial research to close the gap between the two scaling regimes.

Motivated by the robust scaling properties exhibited by other hierarchical systems, this paper describes a gossip algorithm that achieves nearly-optimal performance in wireless networks. We partition the network computation hierarchically in k different scales. At each scale, nodes gossip within their partition until convergence; then one representative is elected within each partition, and the representatives gossip. This is repeated, with representatives gossiping at each higher scale, until the representatives at the coarsest scale have computed x_{ave}. At that point the average is disseminated throughout the network.

The main contribution of this work is a new multi-scale gossip algorithm for which we prove that the average number of messages per node needed to reach average consensus in a grid or random geometric graph on n nodes, scales as $(k+1)n^{\delta(k)}$, where $\delta(k) \to 0$ as $k \to \infty$. Since larger graphs facilitate deeper hierarchies, we show that we can take $k = O(\log\log n)$ to obtain a scheme that asymptotically requires a number of messages per node proportional to $\log\log n$. In simulations of networks with thousands of nodes, two or three levels of hierarchy suffice to achieve state-of-the-art performance. This is comparable to existing state-of-the-art gossip algorithms. Similar to the existing state-of-the-art, we assume the network implements geographic routing. This facilitates gossip iterations between pairs of representatives that may not communicate directly at higher levels of the hierarchy. However, unlike other fast gossip algorithms, we do not require many gossip exchanges among nodes. For example, the path averaging algorithm described in [4] gossips on average among $|S(t)| \propto \sqrt{n}$ nodes along a path at each iteration. In contrast, all gossip iterations in multi-scale gossip are between pairs of nodes, so if a message is lost in transit through the network, the amount of information lost is minimal. Moreover, the longest distance individual messages must travel is on the order of $n^{1/3}$ hops.

2 Previous Work and Known Results

Our primary measure of performance is *communication cost*—the number of messages (transmissions over a single hop) required to compute an estimate to

ϵ accuracy—which is also considered in [4,5]. In the analysis of scaling laws for gossip algorithms, a commonly studied measure of convergence rate is the ϵ averaging time, denoted $T_\epsilon(n)$, which is the number of iterations required to reach an estimate with ϵ accuracy with high probability[1]. $T_\epsilon(n)$ reflects the idea that the complexity of gossiping on a particular class of network topologies should depend both on the final accuracy and the network size. When only neighbouring nodes communicate at each iteration, $T_\epsilon(n)$ and communication cost are identical up to a constant factor. Otherwise, communication cost can generally be bounded by the product of $T_\epsilon(n)$ and a bound on the number of messages required per iteration.

Kempe, Dobra, and Gehrke [2] initiated the study of scaling laws for gossip algorithms and showed that gossip requires $\Theta(n \log \epsilon^{-1})$ total messages to converge on complete graphs. Note that at least n messages are required to compute a function of n distributed data values in any network topology.

In wireless sensor network applications, *random geometric graphs* are a typical model for connectivity since communication is restricted to nearby nodes. In a 2-dimensional random geometric graph, n nodes are randomly assigned coordinates uniformly in unit square, and two nodes are connected with an edge when their Euclidean distance is less than or equal to a connectivity radius, r [6,7]. In [6] it is shown that if the connectivity radius scales as $r_{\text{con}}(n) = \Theta(\sqrt{\frac{\log n}{n}})$ then the network is connected with high probability. Throughout this paper when we refer to a random geometric graph, we mean one with the connectivity $r_{\text{con}}(n)$. Boyd, Ghosh, Prabhakar, and Shah [3] studied scaling laws for pairwise randomized gossip on random geometric graphs and found that communication cost scales as $\Theta(\frac{n^2}{\log n} \log \epsilon^{-1})$ messages even if the algorithm is optimized with respect to the topology.

One approach for improving convergence rates is to introduce memory at each node, creating higher-order updates [8,9]. However, the only known scaling laws for this approach are for a deterministic, synchronous variant of gossip [10], leading to $\Theta(\frac{n^{1.5}}{\sqrt{\log n}} \log \epsilon^{-1})$ communication cost. Gossip algorithms based on lifted Markov chains have been proposed that achieve similar scaling laws [11,12].

A variant called *geographic gossip*, proposed by Dimakis, Sarwate, and Wainwright [5], achieves a similar communication cost of $\Theta(\frac{n^{1.5}}{\sqrt{\log n}} \log \epsilon^{-1})$ by allowing distant (non-neighbouring) pairs of nodes to gossip at each iteration. Assuming that each node knows its own coordinates and the coordinates of its neighbours in the unit square, communication between arbitrary pairs of nodes is made possible using *greedy geographic routing.* Rather than addressing nodes directly, a message is sent to a randomly chosen target (x, y)-location, and the recipient of the message is the node closest to that target. To reach the target, a message is forwarded from a node to its neighbour who is closest to the target. If a node is closer to the target than all of its neighbours, this is the final message recipient. It is shown in [5] that for random geometric graphs with connectivity radius $r(n) = r_{\text{con}}(n)$, greedy geographic routing succeeds with high probability. For an

[1] A more rigorous definition is provided in the next section.

alternative form of greedy geographic routing, which may be useful in implementations, see [13]. The main contribution of [5] is to illustrate that allowing nodes to gossip over multiple hops can lead to significant improvements in communication cost. In follow-up work, Benezit, Dimakis, Thiran, and Vetterli [4] showed that a modified version of geographic gossip, called *path averaging*, can achieve $\Theta(n \log \epsilon^{-1})$ communication cost on random geometric graphs. To do this, all nodes along the path from the source to the target participate in a gossip iteration. If geographic routing finds a path of nodes $S = \{x_i, \ldots, x_j\}$ to deliver a message from x_i to x_j, on the way to x_j values of nodes in S are accumulated. Then x_j computes the average of all S values and sends the average back down the same path towards x_i. All nodes along the way update their values.

The multi-scale approach considered in this paper also assumes that the network is capable of geographic routing in order to gossip among representative nodes at each scale. Below, we show that asymptotically, the communication complexity of multi-scale gossip is $O(n \log \log n \log \epsilon^{-1})$ messages, which is equivalent to that of path averaging up to a logarithmic factor. However, in multi-scale gossip, information is only exchanged between pairs of nodes, and there is no averaging along paths. We believe that this makes our algorithm more fault tolerant, since each message only carries the information for a pair of nodes. If an adversary wishes to disrupt gossip computation by forcing the network to drop a particular message or by deactivating a node in the middle of an iteration, a substantial amount of information can be lost in path averaging since each iteration involves $O(\sqrt{\frac{n}{\log n}})$ nodes on average. In addition, the longest distance a message travels in our multi-scale approach is $O(n^{1/3})$ hops in comparison to $O(n^{1/2})$ hops for geographic gossip or path averaging.

Finally, we note that we are not the first to propose gossiping in a multi-scale or hierarchical manner. Sarkar et al. [14] describe a hierarchical approach for computing aggregates, including the average. However, because their algorithm uses order and duplicate insensitive synopses to estimate the desired aggregate, the size of each message exchanged between a pair of nodes must scale with the size of the network. Other hierarchical distributed averaging schemes that have been proposed in the literature focus on the synchronous form of gossip, and they do not prove scaling laws for communication cost neither do they provide rules for forming the hierarchy (i.e. assume the hierarchical decomposition is given) [15,16,17].

3 Network Model and Problem Definition

Let $G = (V, E)$ be a random geometric graph [7] of n nodes with connectivity radius $r_{con}(n)$. Each node in G holds an initial real value $x_i(0)$. Vector $x(t) = [x_1(t), x_2(t), \ldots, x_n(t)]^{\mathrm{T}}$ describes the values on the network at time t. Our goal is for nodes to communicate over graph edges to compute the average $x_{ave} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$. In the end, all nodes should have knowledge of x_{ave}. To measure performance of an averaging algorithm we use the following definition.

Definition 1. *ϵ averaging time $T_\epsilon(n)$. Given any desired accuracy $\epsilon > 0$, the ϵ averaging time is the earliest time at which vector $x(t)$ is ϵ close to the normalized true average with probability greater than $1-\epsilon$:*

$$T_\epsilon(n) = \sup_{x(0)} \inf_{t=0,1,2,\ldots} \left\{ \mathbb{P}\left(\frac{\|x(t) - x_{ave}\|_2}{\|x(0)\|_2} \geq \epsilon \right) \leq \epsilon \right\} \tag{1}$$

Convergence rate generally depends on the initial values $x(0)$ and the definition assumes the worst possible starting point. In practice, to eliminate the possibility of favourable initial conditions when evaluating our algorithm, as initial condition we assign to each node the sum of its geographic coordinates.

Notice that time is measured in discrete time moments until convergence. As explained in [3] this facilitates the analysis of gossip algorithms but does not force the actual implementation to be sequential. Multiple communication events happen in parallel. Finally, for multi-scale gossip we use $T_\epsilon(n)$ since the consensus time characterization of convergence rate defined in [18] is not applicable. Specifically, our averaging scheme proceeds in phases and changes over time in a non-ergodic manner. Moreover, by definition our scheme stops after a finite number of iterations. The approach in [18] is only defined asymptotically as $t \to \infty$ and for averaging schemes that are ergodic.

4 Multi-scale Gossip

Multi-scale gossip performs averaging in a hierarchical manner. At each moment only nodes in the same level of hierarchy are doing computations at a local scale and computation at one level begins after the previous level has finished. By hierarchically decomposing the initial graph into subgraphs, we impose an order in the computation. As shown in the next section, for a specific decomposition it is possible to divide the overall work into a small number of linear sub-problems and thus obtain very close to linear complexity in the size of the network overall.

Assume a random geometric graph $G = (V, E)$ where each node knows its own coordinates in the unit square and the locations of its immediate neighbours. Each node also knows the total number of nodes in the network n, and k, the desired number of hierarchy levels[2]. Figure 1 illustrates an example with $k = 3$. At level 1 the unit square is split into m_1 small cells – denote them C_1 cells. The subgraphs G_1 of G involving nodes inside a single C_1 cell run standard randomized gossip until convergence. Then, each C_1 cell elects a representative node[3] L_1. The representative selection can be randomized or deterministic as explained in Section 7. Generally, not all G_1 graphs have the same number of nodes. For this reason, the value of each representative has to be reweighed proportionally to its graph size. At level 2, the unit square is split into C_2 cells. Each C_2 contains the same number of C_1 cells. The representatives of the C_1 cells

[2] As explained in Section 6, given n, k can be computed automatically.

[3] Note that in level 1 as well as any other level d there are many C_d cells but for simplicity we avoid denoting them $C_{d,i}$ with i running from 1 to maximum number of cells. Similarly for representatives L_d.

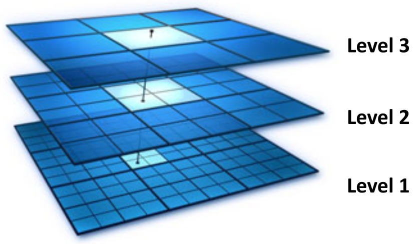


Fig. 1. Hierarchical multiscale subdivision of the unit square. At each level, each cell is split into equal numbers of smaller cells. Before the representatives of the cells can gossip on a grid graph, we run gossip on each cell.

form grid graphs G_2 with two representatives $L_{1,i}$ and $L_{1,j}$ sharing an edge in a G_2 if cells $C_{1,i}$ and $C_{1,j}$ are adjacent and contained in the same C_2 cell. Note that representatives can determine which cells they are adjacent to given the current level of hierarchy and n. Next, we run randomized gossip simultaneously on all G_2 grid graphs. Finally, we select a representative node L_2 out of each G_2 and continue the next hierarchy level. The process repeats until we reach level k at which point we have only one grid graph G_k contained in the single cell C_k which coincides with the unit square. Once gossip on G_k is over, each representative L_{k-1} disseminates his final value to all the nodes in its cell.

Algorithm 1 describes multi-scale gossip in a recursive manner. The initial call to the algorithm has as arguments, the vector of initial node values (x_{init}), the unit square ($C = [0,1] \times [0,1]$), the network size n, the desired number of hierarchy levels k and the desired error tolerance *tol*. In a down-pass the unit square is split into smaller and smaller cells all the way to the C_1 cells. After gossiping in the G_1 graphs in *Line 15*, the representatives adjust their values (*Line 16*). As explained in the next section, if k is large enough, the G_1 are complete graphs. Since each node knows the locations of its immediate neighbours (needed for geographic routing), at level 1 it is easy to also compute the size of each G_1 which is needed for the reweighting. The up-pass begins with the L_1 representatives forming the G_2 grid graphs (*Line 8*) and then running gossip in all of them in parallel.. We use a parameter $a = \frac{2}{3}$ to decide how many C_{d-1} cells fit in each C_d cell. The motivation for this parameter and its specific value are explained in the following section. Notice the pseudocode mimics a sequential single processor execution which is in line with the analysis that follows in Section 6. However, it should be emphasized that the algorithm is intended for and can be implemented in a distributed fashion. The notation $x_{init}(C)$ or $x_{init}(L)$ indicates that we only select the entries of x_{init} corresponding to nodes in cell C or representatives L.

The ideal scenario for multi-scale gossip is if computation inside each cell stops automatically when the desired accuracy is reached. This way no messages are wasted. However in practice cells at the same level we may need to gossip

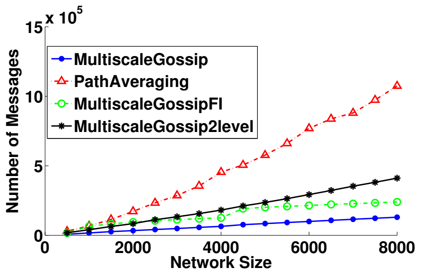
Algorithm 1. MultiscaleGossip(x_{init}, C, n, k, tol)

```
1: $a = \frac{2}{3}$
2: **if** $k > 1$ **then**
3:     Split $C$ into $m_{k-1} = n^{1-a}$ cells: $C_{k-1,1}, \ldots, C_{k-1,m_{k-1}}$
4:     Select a representative node $L_{k-1,i}$ for each cell $C_{k-1,i}, i \in \{1, \ldots, m_{k-1}\}$
5:     **for all** cells $C_{k-1,i}$ **do**
6:         **call** *HierarchicalGossip*($x_{init}(C_{k-1,i}), C_{k-1,i}, n^a, k-1, tol$)
7:     **end for**
8:     Form grid graph $G_{k-1}$ of representatives $L_{k-1,i}$
9:     **call** *RandomizedGossip*($x_{init}(L_{k-1,1:m_{k-1}}), G_{k-1}, tol$)
10:    **if** at top level **then**
11:        Spread value of $L_{k,i}$ to all nodes in $C_{k,i}$
12:    **end if**
13: **else**
14:    Form graph $G_1$ only of nodes in $V(G)$ contained in $C$
15:    **call** *RandomizedGossip*($x_{init}, G_1, tol$)
16:    Reweight representative values as : $x(L_{1,i}) = x(L_{1,i})\frac{|V(G_1)| \cdot m_2}{|V(G)|}$
17: **end if**
```

on graphs of different sizes that take different numbers of messages to converge. This creates a need for node synchronization so that all computation in one level is finished before the next level can begin. To overcome the need for synchronization, we can fix the number of randomized gossip iterations per level. Given that nodes are deployed uniformly at random in the unit square, we can make a worst case estimate of how many nodes are expected to be in a cell of a certain area. Since by construction all cells at the same level have equal area, we gossip on all graphs at that level for a fixed number of iterations. Usually, at level 1, we have less nodes than expected so we end up wasting messages running gossip for longer than necessary.

5 Evaluation of Multi-scale Gossip

Before proceeding with the formal analysis of the algorithm complexity, we show in this section that Multi-scale Gossip performs very well against Path Averaging [4], a recent state-of-the-art gossip algorithm that requires linear number messages in the size of the network to converge to the average with ϵ accuracy. Figure 2 (left) shows the number of messages needed to converge within $\epsilon = 0.0001$ error for graphs of sizes 500 to 8000. The bottom curve tagged *MultiscaleGossip* shows the ideal case where computation inside each cell stops automatically when the desired accuracy is reached. The curve tagged *MultiscaleGossipFI* was generated using fixed number of iterations per level based on worst case graph sizes as explained in Section 4. One reason why path averaging seems to be slower is because we use a smaller connectivity radius for our graphs ($r = \sqrt{\frac{3 \log n}{n}}$ instead of $r = \sqrt{\frac{10 \log n}{n}}$ which is described in [4]).



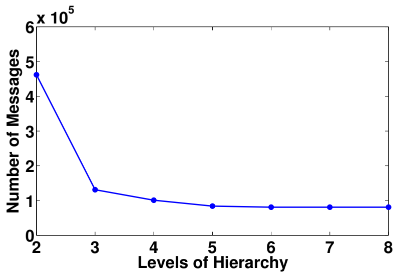


Fig. 2. (left) Comparison of MultiscaleGossip to PathAveraging. Total number of messages to converge with $\epsilon = 0.0001$ accuracy on random geometric graphs of increasing sizes. Results are averages over 20 runs. MultiscaleGossip used with 5 levels of hierarchy. MultiscaleGossipFI is the version using a fixed number of iterations for gossiping at a specific level. MultiscaleGossip2level is a version using only two levels of Hierarchy and is explained in Section 7. (right) Increasing the levels of Hierarchy yields a diminishing reward. Results are averages over 10 random geometric graphs with 5000 nodes and final desired accuracy $\epsilon = 0.0001$. All graphs are created with radius $r = \sqrt{\frac{3 \log n}{n}}$.

Multi-scale Gossip has several advantages over Path Averaging. All the information relies on pairwise messages. In contrast, averaging over paths of length more than two has two main disadvantages. First, if a message is lost, a large number of nodes (potentially $O(\sqrt{\frac{n}{\log n}})$) are affected by the information loss. Second, when messages are sent to a remote location over many hops, they increase in size as the message body accumulates the information of all the intermediate nodes. Besides being variable, the message size now depends on the length of the path and ultimately on the network size. Our messages are always of constant size and independent of the hop distance or network size. Moreover, as will be shown in the next section, the maximum number of hops any message has to travel is $O(n^{\frac{1}{3}})$ at worst. This should be compared to distance $O(\sqrt{n})$ which is necessary for Path Averaging to achieve linear scaling. Finally, Multi-scale Gossip is relatively easy to analyze and implement using standard randomized gossip as a building block for the averaging computations.

6 Analysis of Multi-scale Gossip

The motivation behind multi-scale gossip is to divide the computation into stages each of which takes no more than linear number of messages in the size of the network. This allows the overall algorithm to scale very close to linear as established by the following theorem:

Theorem 1. *Let a random geometric graph G of size n and an $\epsilon > 0$ be given. As the graph size $n \to \infty$, the communication cost of the multi-scale gossip scheme described above with scaling constant $\alpha = \frac{2}{3}$ behaves as follows:*

1. *If the number of hierarchy levels k remains fixed as $n \to \infty$, then the communication cost of multi-scale gossip is $O\big((kn + n^{1+(\frac{2}{3})^k}) \log \epsilon^{-1}\big)$ messages.*
2. *If $k = \Theta(\log\log n)$, then the communication cost of multi-scale gossip is $O(n \log\log m \log \epsilon^{-1})$ messages.*

Proof. Suppose we run Multi-scale Gossip (Algorithm 1) on a random geometric graph $G = (V, E)$ with $|V| = n$ and connecting radius is $r(n) = \sqrt{\frac{c \log n}{n}}$. Call the unit square cell C_k for a total of k hierarchy levels. At the highest level, we split the unit square into $m_{k-1} = n^{1-a}$ cells $C_{k-1,i}$ each of dimensions $n^{\frac{a-1}{2}} \times n^{\frac{a-1}{2}}$ where $a = \frac{2}{3}$ as explained below. On a grid of p nodes, randomized gossip requires $O(p^2)$ or by including the dependence on final accuracy, $O(p^2 \log \epsilon^{-1})$ messages to converge (e.g. see [3]). The grid graph G_{k-1} formed by the representatives of the $C_{k-1,i}$ cells has n^{1-a} nodes. Moreover, for appropriately large c (e.g. $c = 3$), graph G is *geo-dense* [19] and a patch of area n^{a-1} is expected to have $\Theta(n^{a-1}n) = \Theta(n^a)$ nodes in it. The maximum distance between two representatives in G_{k-1} will be $\sqrt{5}n^{\frac{a-1}{2}} = O(n^{\frac{a-1}{2}})$. If we divide by $r(n)$, we get a worst case estimate of the cost for multi-hop messages between representatives: $MsgCost_{k-1} = O(n^{\frac{a}{2}})$. Now the total number of pairwise messages on G_{k-1} will be $O((n^{1-a})^2 \log \epsilon^{-1}) \cdot O(n^{\frac{a}{2}})$. This number is $O(n)$ if $a = \frac{2}{3}$.

At the next level of hierarchy, we subdivide each $C_{k-1,i}$ cell of $q = \Theta(n^a)$ nodes into $q^{1-a'}$ cells in a recursive manner. Each $C_{k-2,i}$ cell will have dimensions $q^{\frac{a'-1}{2}} \times q^{\frac{a'-1}{2}}$. Using the exact same analysis as above, we will have n^{1-a} grid graphs GG_{k-2} and each has $q^{1-a'}$ (representative) nodes. The communication cost between $L_{k-2,i}$ representatives is $MsgCost_{k-2} = O(q^{\frac{a'}{2}})$. To make the total number of messages at level $k-2$ linear, we get $n^{1-a} \cdot O((q^{1-a'})^2 \log \epsilon^{-1}) \cdot O(q^{\frac{a'}{2}}) = n \Rightarrow a' = \frac{2}{3}$ as well. A simple induction proves that in general a subdivision of each cell of size q into q^{1-a} subcells yields linear performance for that level if $a = \frac{2}{3}$. Finally, after k levels, the algorithm runs gossip on each subgraph of G with nodes contained inside each of the C_1 cells. We have $O(n^{1-(\frac{2}{3})^k})$ C_1 cells, each containing $n^{(\frac{2}{3})^k}$ nodes. Since we run randomized gossip on each subgraph, the total number of messages at the last level is $O(n^{1+(\frac{2}{3})^k})$. Summing up all levels, plus n messages to spread the final result back to all nodes,the total number of messages for Multi-scale Gossip is $O\big((kn + n^{1+(\frac{2}{3})^k}) \log \epsilon^{-1}) + n\big) = O\big((kn + n^{1+(\frac{2}{3})^k}) \log \epsilon^{-1})\big)$.

For the second part of the theorem, observe that at level 1 each cell contains a subgraph of $n^{(\frac{2}{3})^k}$ nodes in expectation. We can choose k so that each cell contains at least $m \geq 2$ nodes for randomized gossip to be non-trivial and no more than $M \geq m$ nodes so that the cost per cell is bounded by $M^2 \log \epsilon^{-1}$. In other words, choose k such that:

$$m \leq n^{(\frac{2}{3})^k} \leq M \Rightarrow \frac{\log\log M - \log\log n}{\log \frac{2}{3}} \leq k \leq \frac{\log\log m - \log\log n}{\log \frac{2}{3}}$$

Since the cost per level 1 cell is now bounded by a constant for $k = \Theta(\log\log n)$, the total level 1 cost is $O(n^{1-(\frac{2}{3})^k} \log \epsilon^{-1})$ and the overall cost is $O\big((kn + n^{1-(\frac{2}{3})^k}) \log \epsilon^{-1}) + n\big) = O(n \log\log n\epsilon^{-1})$ □

In practice we only need a few levels of hierarchy. Figure 2 (right) investigates the effect of increasing the levels of hierarchy. The figure shows the number of messages until convergence within 0.0001 error, averaged over ten graphs of 5000 nodes. More levels yield a diminishing reward and we don't need more than 4 or 5 levels. As discussed in Section 7 these observation lead us to try a scheme with only two levels of hierarchy which still produces an efficient algorithm.

7 Practical Considerations

There is a number of practical considerations that we would like to bring to the reader's attention. We list them in the form of questions below:

Does Multi-scale gossip computation scheme affect the final error? This is a valid concern since our algorithm essentially uses randomized gossip as a lossy averaging operator over subsets of the network nodes. At each level the representatives trust the ϵ approximate values of the previous level. Fortunately the error deteriorates only linearly with the number of levels. If $ave(\cdot)$ is an ϵ accuracy averaging operator, $ave(a_1, \dots a_m) = \overline{a} \pm \epsilon$. At the next level, $ave(\overline{a}_1 \pm \epsilon, \dots, \overline{a}_m \pm \epsilon) = \overline{a} \pm \epsilon \pm \epsilon$. After k levels the final error will be at worst $\pm k\epsilon$.

How can we detect convergence in a subgraph or cluster? Do the nodes need to be synchronized? At each hierarchy level, representatives know how big the grid that they are gossiping over is (function of n and k only). Moreover, all grids at the same level are of the same size and we have tight bounds on the number of messages needed to obtain ϵ accuracy on grids w.h.p. We can thus gossip on all grids for a fixed number or rounds and synchronization is implicit. At level 1 however, in general we need to gossip on random geometric subgraphs which are not of exactly the same size. As n gets large though, random geometric graphs tend to become regular and uniformly spaced on the unit square. Therefore, the subgraphs contained in cells at level 1 all have sizes very close to the expected value of $n^{(\frac{2}{3})^k}$. Thus, we run gossip for a fixed number of rounds using the theoretical bound for graphs of the size $n^{(\frac{2}{3})^k}$. As discussed in Section 4, fixing the number of iterations leads to redundant transmissions, however the algorithm is still very efficient.

What happens with disconnected subgraphs or grids due to empty grid cells? Technically this is possible since the division of the unit square into grid cells does not mean that each cell is guaranteed to contain any nodes of the initial graph. Representatives use multi-hop communication and connected grids can always be constructed as long as the initial random geometric graph is connected. At level 1 the subgraphs of the initial graph contained in each cell could still be disconnected if edges that go outside the cell are not allowed. However, as explained in Section 6 we can use enough hierarchy levels so that each C_1 cell is a complete graph and the probability of getting disconnected C_1 cells tends to zero.

How can we select representatives in a natural way? The easiest solution is to pick the point p_c that is geographically at the centre of each cell. Again, knowledge of n, k uniquely identifies the position of each cell and also p_c. By sending all messages to p_c, geographic routing will deliver them to the unique node that is closest to that location w.h.p. To change representatives, we can deterministically pick a location $p_c + u$ which will cause a new node to be the closest to that location. A more sophisticated solution would be to employ a randomized auction mechanism. Each node in a cell generates a random number and the largest number is the representative. Once a new message enters a cell, the nodes knowing their neighbours values, route the message to the cell representative. Notice that determining cell leaders this way does not incur more than linear cost.

Are representatives bottlenecks and single points of failure? This is not an issue. There might be a small imbalance in the amount of work done by each node, but it can be alleviated by selecting different representatives at each hierarchy level. Moreover, for increased robustness, at a linear cost we can disseminate the representative's values to all the nodes in its cell. This way if a representative dies, another node in the cell can take its place. The new representative will have a value very similar (within ϵ) to that of the initial representative at the beginning of the computation at the current level. Thus node failure is expected to only cause small delay in convergence at that level. We should emphasize however that the effect of node failures has received little attention so far and still asks for a more systematic investigation.

How much extra energy do the representatives need to spend? This question is hard to answer analytically. We use simulation to get a feel for it. Figure 3 shows the number of messages sent by each of the 5000 nodes in a random geometric graph. For this case we used five levels of hierarchy. The first 3200 nodes were representatives at some point in the computation and the rest only participated in gossiping at level 1. As we go down the hierarchy the cells get smaller and the options for representatives are less so it is expected that some nodes will be doing more computation. Here the average number of messages per representative is about 16 with a standard deviation of 14. However, only less than 10% of the representative nodes send more than 30 messages. For the level 1, nodes have an average of 5 messages with standard deviation of 3.

Don't many levels of hierarchy make it harder to implement and keep the nodes synchronized? This is a valid concern. Given our observations in Section 6 we tried an algorithm with just 2 levels of hierarchy. In this case, for graphs of size a few thousand nodes, splitting the unit square into n^{1-a} cells with

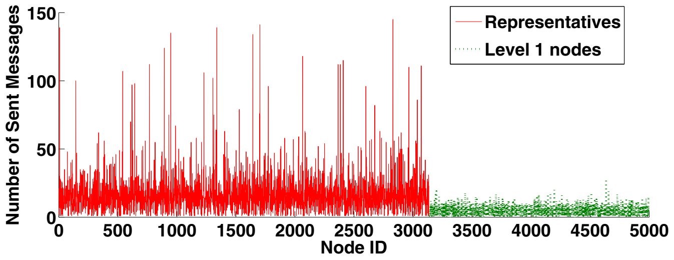


Fig. 3. Node utilization on a random geometric graph with 5000 nodes and final desired accuracy $\epsilon = 0.0001$. For each node we plot the total number of sent messages. Nodes 0 to 3200 have had at least one representative node role and the rest only participated at level 1. Both types of nodes may have participated as indermediates in multi-hop communication as well.

$a = \frac{2}{3}$ is not a good choice as it produces a very small grid of representatives and quite large level 1 cells. To achieve better load balancing between the two levels, we use $a = \frac{1}{2}$. This choice has the advantage that the maximum number of hops any message has to travel is $O(n^{\frac{1}{4}})$. To see this, observe that each cell C_1 has area $\frac{1}{n^a} = n^{-\frac{1}{2}} = n^{-\frac{1}{4}} \times n^{-\frac{1}{4}}$. Thus the maximum distance between representatves is $O(n^{-\frac{1}{4}})$. If we divide by the connecting radius $r(n) = \sqrt{\frac{c \log n}{n}}$ we get the result. Another interesting finding is that for moderate sized graphs, using cells of area $n^{-\frac{1}{2}}$ produces subgraphs which are very well connected. Since nodes are deployed uniformly at random, an area $n^{-\frac{1}{2}}$ is expected to contain $n^{\frac{1}{2}}$ nodes. A subgraph inside a C_1 cell is still a random geometric graph with $t = n^{\frac{1}{2}}$ nodes, but for which the radius used to connect nodes is not $\sqrt{\frac{c \log t}{t}}$. It is $\sqrt{\frac{c \log n}{n}}$. This is equivalent to creating a random geometric graph of t nodes in the unit square but with a scaled up radius of $r_t = \sqrt{\frac{c \log n}{t}}$. From [19] we know that a random geometric graph of t nodes is rapidly mixing (i.e. linear number of messages for convergence) if the connecting radius is $r_{rapid} = \frac{1}{poly(\log t)}$. Now, e.g. for $c = 3$ and $n \leq 9 * 10^6$, we get $r_t \geq \frac{1}{\log t} \geq r_{rapid}$ for $t = \sqrt{n} \leq 3000$. Consequently, the C_1 cells are rapidly mixing for networks of less than a few millions of sensors. In Figure 2 verifies this analysis. For graphs from 500 to 8000 nodes and final error 0.0001, we see that *MultiscaleGossip2level* performs very close to Multi-scale Gossip with more levels of hierarchy and better than path averaging.

8 Discussion and Future Work

We have presented a new algorithm for distributed averaging exploiting hierarchical computation. Multi-scale gossip separates the computation in linear phases and achieves very close to linear complexity overall ($O((k+1)n^{1+\delta(k)})$). Moreover, the maximum distance any message has to travel is $O(n^{\frac{1}{3}})$ as opposed to $O(\sqrt{n})$ needed by path averaging which is the other existing gossip algorithm with linear complexity. Finally, multi-scale gossip uses fixed size messages independent of the graph size and does not rely on longer that pairwise averaging operations. There is a number of interesting future directions that we see. In our present description, computation happens on grids which are known to require quadratic number of messages. Since these grids use multi-hop communication anyway, it might be possible to further increase performance by devising other overlay graphs between representatives with better convergence properties, i.e. expander graphs [20]. Moreover, the subdivision of the unit square into grid cells is not necessarily natural with respect to the topology of the graph. One could use other methods for clustering. So far, we have some preliminary results with spectral clustering which seem promising in simulation. It is however not clear how to do spectral clustering in a distributed way and in linear number of messages. Another idea is to combine the multi-scale approach with the use of more memory at each node to get faster mixing rates. Notice however that how to use memory to provably accelerate asynchronous gossip is still an open question. Current results only look at synchronous algorithms [10]. Finally, an important advantage of gossip algorithms in general is their robustness. Intuitively this is expected. However the question of modelling and reacting to node failures has not been formally investigated in the literature. It would be very interesting to introduce failures and see the effect on performance for different gossip algorithms.

Acknoledgements

We would like to thank Marius Şucan for providing the multiscale grid figure in Section 4.

References

1. Tsitsiklis, J.: Problems in Decentralized Decision Making and Computation. PhD thesis, Massachusetts Institute of Tech. (November 1984)
2. Kempe, D., Dobra, A., Gehrke, J.: Computing aggregate information using gossip. In: Proc. Foundations of Computer Science, Cambridge, MA (October 2003)
3. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Trans. Inf. Theory 52(6), 2508–2530 (2006)
4. Benezit, F., Dimakis, A., Thiran, P., Vetterli, M.: Gossip along the way: Order-optimal consensus through randomized path averaging. In: Proc. Allerton Conf. on Comm., Control and Comp., Urbana-Champaign, IL (September 2007)
5. Dimakis, A., Sarwate, A., Wainwright, M.: Geographic gossip: Efficient averaging for sensor networks. IEEE Trans. Signal Processing 56(3), 1205–1216 (2008)
6. Gupta, P., Kumar, P.R.: Critical power for asymptotic connectivity in wireless networks. In: Stochastic Analysis, Control, Optimization and Applications, Boston, pp. 1106–1110 (1998)

7. Penrose, M.: Random Geometric Graphs. Oxford University Press, Oxford (2003)
8. Cao, M., Spielman, D.A., Yeh, E.M.: Accelerated gossip algorithms for distributed computation. In: Proc. 44th Annual Allerton Conf. Comm., Control and Comp., Monticello, IL (September 2006)
9. Kokiopoulou, E., Frossard, P.: Polynomial filtering for fast convergence in distributed consensus. IEEE Trans. Signal Processing 57(1), 342–354 (2009)
10. Oreshkin, B., Coates, M., Rabbat, M.: Optimization and analysis of distributed averaging with short node memory. To appear IEEE Trans. Signal Processing (2010)
11. Li, W., Dai, H.: Location-aided fast distributed consensus. IEEE Transactions on Information Theory (2008) (submitted)
12. Jung, K., Shah, D., Shin, J.: Fast gossip through lifted Markov chains. In: Proc. Allerton Conf. on Comm., Control and Comp., Urbana-Champaign, IL (September 2007)
13. Sarkar, R., Yin, X., Gao, J., Luo, F., Gu, X.D.: Greedy routing with guaranteed delivery using ricci flows. In: Proc. Information Processing in Sensor Networks, San Francisco (April 2009)
14. Sarkar, R., Zhu, X., Gao, J.: Hierarchical spatial gossip for multi-resolution representations in sensor networks. In: Proc. of the International Conference on Information Processing in Sensor Networks (IPSN 2007), April 2007, pp. 420–429 (2007)
15. Kim, J.H., West, M., Lall, S., Scholte, E., Banaszuk, A.: Stochastic multiscale approaches to consensus problems. In: Proc. IEEE Conf. on Decision and Control, Cancun (December 2008)
16. Epstein, M., Lynch, K., Johansson, K., Murray, R.: Using hierarchical decomposition to speed up average consensus. In: Proc. IFAC World Congress, Seoul (July 2008)
17. Cattivelli, F., Sayed, A.: Hierarchical diffusion algorithms for distributed estimation. In: Proc. IEEE Workshop on Statistical Signal Processing, Wales (August 2009)
18. Denantes, P., Benezit, F., Thiran, P., Vetterli, M.: Which distributed averaging algorithm should i choose for my sensor network? In: Proc. IEEE Infocom, Phoenix (April 2008)
19. Avin, C., Ercal, G.: On the cover time and mixing time of random geometric graphs. Theoretical Computer Science 380, 2–22 (2007)
20. Margulis, G.: Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. J. Probl. Inf. Transm. 24(1), 39–46 (1988)

Wormholes No More? Localized Wormhole Detection and Prevention in Wireless Networks

Tassos Dimitriou and Athanassios Giannetsos

Athens Information Technology,
19002, Athens, Greece
{tdim,agia}@ait.edu.gr

Abstract. A number of protocols have been proposed to date to defend against *wormhole attacks* in wireless networks by adopting synchronized clocks, positioning devices, or directional antennas. In this work, we introduce a novel approach for detecting wormhole attacks. The proposed algorithm is completely *localized* and works by looking for simple evidence that no attack is taking place, using only connectivity information as implied by the underlying communication graph, and total absence of coordination. Unlike many existing techniques, it does not use any specialized hardware, making it extremely useful for real-world scenarios. Most importantly, however, the algorithm can *always* prevent wormholes, irrespective of the density of the network, while its efficiency is not affected even by frequent connectivity changes. We also provide an analytical evaluation of the algorithm's correctness along with an implementation on real sensor devices that demonstrates its efficiency in terms of memory requirements and processing overhead.

Keywords: Wormhole attack, Sensor Networks, Ad hoc networks, Computer network security, Tunnelling, Connectivity Information, Path Existence.

1 Introduction

The use of wireless ad hoc and sensor networks has given birth to a broad class of exciting new applications in several areas of our lives. However, these networks are exposed to security threats which, if not properly addressed, can exclude them from being deployed in the envisaged scenarios.

The *wormhole attack* [7] is a severe threat against wireless networks in which an adversary tunnels messages received in one part of the network and replays them in a different part, as shown in Figure 1(a). Once the wormhole link is operational, the adversary eavesdrops messages at one end and forwards them (possibly selectively) to the other end, where the packets are retransmitted. Thus nodes who would normally be multiple hops away from a sink are convinced that they are only one or two hops away via the wormhole.

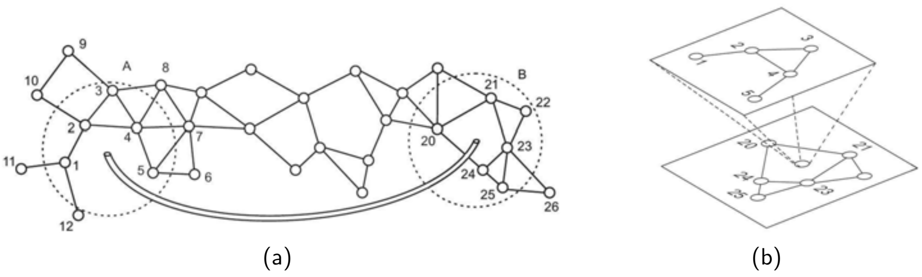


Fig. 1. (a) Demonstration of a wormhole attack. Nodes in area A consider nodes in area B their neighbors and vice versa. (b) Projection of the two ends of the wormhole.

The net effect of the wormhole attack is that nodes within region A think they are neighbors with nodes within region B and vice versa. This is equivalent with taking all nodes in one area and place them at just one point within another area, as shown in Figure 1(b). Thus an attacker can use it to completely disrupt routing and attract a significant amount of traffic enabling other kind of attacks such as the Sinkhole attack [7,10].

Our contribution: In this paper, we explore the development of a *localized* algorithm that can detect wormhole attacks on wireless networks directly based on *connectivity* information implied by the underlying communication graph. Our work deviates from the customary strategies of using specialized hardware in sensors, directional antennas, tight clock synchronization, or distance measurements between the nodes, thus making this approach universally applicable.

The detection algorithm searches for simple structures that indicate that no attack is taking place and its efficiency is not affected even by frequent connectivity changes, another deviation from past works that assume *static* topologies. We provide an analytical evaluation of the algorithm's performance and correctness showing that attack prevention is 100% (even for low density networks) while keeping the running time and the percentage of false positives very small. Finally, we present an implementation of the algorithm on real sensor devices, justifying its efficiency in terms of memory requirements and processing overhead. In general, we expect that our approach will have a practical use in sensor network deployments overcoming some of the limitations of existing techniques.

Paper Organization: The remainder of this paper is organized as follows. First, we discuss related work in Section 2 and state our assumptions in Section 3. Section 4 is the heart of this work; it gives insight on the algorithm, a mathematical proof about its correctness in preventing wormhole attacks and a probabilistic analysis on its behavior on legitimate node addition. Performance evaluation, in terms of simulations and experiments on real sensor devices, is presented in Section 5. Section 6 sets forth some attributes of our algorithm for discussion, while Section 7 concludes the paper.

2 Related Work

In this section we briefly describe the most widely known proposed measures for mitigating the effects of a wormhole attack in wireless networks. Enhancing routing with strong node authentication and lightweight cryptography [7] was proposed as a countermeasure, however, the wormhole attack still cannot be defeated that easily as an attacker can simply forward existing packets.

An alternative set of mechanisms that operate independently from the underlying routing protocol is based on *distance/time bounding* techniques. In [5], the authors add a secure constraint (leash) to each packet such as timing or location information that used to tell whether a packet has traveled a distance larger than physically possible. This technique works fine in the presence of specialized hardware for localization and synchronization, however this assumption raises questions about its applicability to ordinary sensor networks.

Another set of solutions use authenticated *distance bounding* of the round trip time (RTT) of a message [3,8,15], received signal strength, or time difference of arrival [16] in order to ensure that nodes are close to each other. The effectiveness of such schemes relies on the immediate reception of responses to challenges sent, however, this may not be possible as MAC protocols introduce random delays between the time a packet is sent and the actual time it is transmitted via the radio interface. In [14] the authors proposed a solution that assumes the existence of a small fraction of nodes that are aware of their own location. This approach is showed to successfully detect a wormhole with probability close to one. However, its functionality relies on the correct operation of distinguished guard nodes that can become single points of failure.

Another line of defense is the use of *graph theoretic and visualization* approaches. In [17], a centralized detection technique is proposed which uses distance estimations between neighboring nodes in order to determine a "network layout" and identify inconsistencies in it, using multi-dimensional scaling (MDS) techniques. However, its centralized nature limits its applicability in sensor networks. Similarly the works in [14,6] make use of guard nodes that attest the source of each transmission. However, these techniques either make location claims or use special purpose hardware making these approaches impractical.

Finally, another interesting approach to date is the work in [13] which looks for topological violations in the underlying connectivity graph in order to detect a wormhole attack. For example, in Figure 1(b), nodes 1, 3 and 5 cannot possibly be neighbors of (say) 20 and 23 since it can be shown that three 2-hop neighbors cannot be the common neighbors of two other nodes u and v. This is an instance of *forbidden substructure* and this is exactly the approach followed in [13] for detecting a wormhole.

While this is a nice, localized approach that uses connectivity information to detect a wormhole attack it suffers from a number of shortcomings. First, it does not always guarantee detection since existence of three 2-hop neighbors lying in the common intersection area of two others depends on the density of the network. The technique will probably fail when the average neighborhood size is low. To alleviate this problem one may look for similar forbidden substructures of size f_k among k-hop neighbors of u and v. Unfortunately, there are two issues with this approach. The first one is that the forbidden substructure is really an *independent set* of the set of common $2k$-hop neighbors $C_{2k}(u,v)$, a known NP-complete problem. Of course one may try to find a maximal independent set in $C_{2k}(u,v)$ and try to compare it with f_k. If the size of the independent set is equal or greater than f_k then an alarm can be raised. However, the value of f_k for $k > 1$ cannot be estimated that easily. For example, for $k = 2$ (2-hop case) this number is as big as 19 [13]. Unless the node density is extremely high, it is unlikely that one will be able to find that many common independent 2-hop neighbors in order to detect the attack.

3 System Model and Assumptions

3.1 Sensor Nodes and Communication

In our model, a wireless sensor network consists of a set $S = \{s_1, s_2, ..., s_n\}$ of n sensor nodes. Sensor nodes are considered neighbors when the distance between them is shorter than some range r. For any sensor node s, the set of neighboring nodes is denoted by $N(s)$. Our first assumption concerns neighborhood information and is denoted by SMA-1 (for System Model Assumption):

SMA-1. All sensors run some neighbor discovery routine, as part of their routing protocol (e.g. using regular route updates), and they can record their neighbor IDs. Thus every node knows its k-hop neighborhood, where k is a small number, typically 1 or 2.

The above assumption is light and realistic, considering the case of sensor networks with frequent neighborhood changes[1]. Furthermore, a lot of research in such networks has been conducted based on the 2-hop neighborhood knowledge, covering areas from energy consumption efficiency [11,12] to intrusion detection [4,9]. In any case, as we will see shortly and discuss further in Section 6, even if SMA-1 does not hold, *no wormholes can be allowed in the network.*

3.2 Attacker Model

Consider an adversary that tries to mound a wormhole attack. The wormhole is a dedicated connection between two physical locations, called *wormhole endpoints.* By re-transmitting packets, an adversary can have nodes hear each other and establish a neighbor relationship, even if they do not reside in each other's radio range. Below is what we have assumed to better model the wormhole attack:

AMA-1. In this work, we will consider attacks in which the wormhole link is long enough so that regions A and B are well separated from each other [2,19].

[1] Notice that this assumption does not say anything about the *validity* of these neighbor IDs. Only that these IDs are forwarded properly by the routing protocol.

Thus it makes no sense to have overlapping endpoints or endpoints close to each other. In particular, we will assume that the real shortest path distance between the two wormhole regions is bigger than $2k$ hops, where k (usually 1 or 2) denotes the k-hop neighborhood structure known to nodes.

We place no restrictions on what an adversary can do with packets that carry neighborhood information. The adversary can drop these packets, however, even in this case, no wormholes can be allowed in the network. In the sections that follow we will see that both assumptions SMA-1 and AMA-1 are not necessary to the correctness of this work. However, the following assumption is:

AMA-2. There is some initial interval t_Δ where no attack has taken place and nodes have safely established their neighborhood information.

This assumption simply says that there must be some initial interval where the network is safe in order for the algorithm to guarantee prevention of wormhole attacks from that time on. This is a standard assumption made in many of the works in this area [2,13,14,17,18,19] and more general in works where some sort of security infrastructure has to be bootstrapped.

4 Localized Wormhole Detection and Prevention

Our approach is strictly *localized* and looks only at *connectivity* information as implied by the underlying communication graph. However, instead of looking for forbidden substructures as in [13], we look for evidence that no attack is taking place. This has some interesting consequences.

- Our approach is applicable even when the communication model is unknown or the network is deployed in an ad-hoc manner.
- Second, in the case of an attack, the algorithm always prevents the attack.
- Third, our algorithm "fails safe"; in the unlikely case where this desired property is not met, the algorithm treats suspicious nodes as participating in an attack. Thus, no wormhole can ever be established. This is in contrast with [13] where wormhole detection cannot be guaranteed in all cases.
- Finally, as we will see in Section 5, the algorithm is very easy to be implemented, even in resource constrained devices such as sensor nodes.

To understand the workings of the algorithm, let's assume that up to time $t \geq t_\Delta$ the network is "safe" (i.e. no wormholes have occurred – Assumption AMA-2) and nodes know each other's k-neighborhoods (Assumption SMA-1).

At some time $t' > t$ a number of neighboring nodes in a set $U = \{u_1, u_2, \ldots\}$ overhear some packets transmitted that include the IDs of new nodes in a set $V = \{v_1, u_v, \ldots\}$. These packets may or may not be the result of a wormhole attack. For example, they may be *newly added* nodes or simply nodes that awoke from a sleeping phase and participate in the workings of the underlying routing protocol. Or in the case of wormhole attack (Figure 1), they may be retransmissions of packets from one area to another. We call the nodes in V *suspected* nodes.

Each node $u \in U$ must determine for each node $v \in V$ whether it should include v in its neighborhood structure. This is done using the following test.

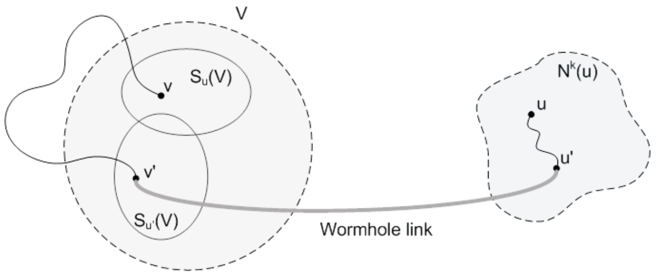


Fig. 2. Contradiction argument

4.1 Local Path Existence Test for Wormhole Prevention

The *localized* test run by each node $u \in U$ for each node $v \in V$ is to determine whether a *small* path (of length no more than $2k$) exists that connects u to v but in a way that *excludes all suspect nodes.* This is possible since u also knows the k-hop neighbors of v. This is done by considering whether u and v have a common neighbor lying in their intersection or two neighbors x and y, respectively, that are either directly connected to each other or have a common 1-hop neighbor z. In the more general case, x and y can either be directly connected or have a common $(k-1)$-hop neighbor. This alternative path, if it exists, will be an attestation that no wormhole link exists and u can safely add v in its neighborhood list. Otherwise v is deleted from the neighborhood of u. This is captured by the following theorem:

Theorem 1. *In case of a wormhole attack, the previous test prevents a wormhole link from being established. If no attack is taking place, the algorithm allows new nodes to be part of the network with high probability.*

Proof. For a formal proof we will consider two cases: i) the case of an actual wormhole attack, and ii) the case of legitimate addition of new nodes.

Case 1: Let $u \in U$ be a node wishing to test whether some suspect node $v \in V$ should be included in its neighborhood list. Let $D > 2k$ be the *real* shortest path distance between u and v in the network (assumption AMA-1 on separation of wormhole endpoints). Let also $N^k(v)$ denote the set of *valid* k-hop neighbors of v also known to u (Assumption AMA-2), and let $S_u(V)$ denote the nodes suspected by u from the set V. *We place no restriction on the suspect set, so neither all nodes in U may suspect the same set of nodes, nor they have to agree on a common suspect list which would raise synchronization issues.*

Consider Figure 2 showing the set of k-hop neighbors of u on the right and the suspect node v on the left. Since u *itself* will check for the existence of a path between u and v of length no more than $2k$, the only way such a path can exist is if the path at some point utilizes the wormhole link and two nodes u' and v' that are at most k hops away from u and v, respectively. If the path from v uses only nodes in $S_u(V)$, u will easily reject such a path. The problem arises when the path uses nodes not in $S_u(V)$. There are two cases here to consider.

The first case is when the path consists of nodes outside V. But in this case u will reject all these paths since their length is at least D and they can never reach u in at most $2k$ hops. The second case is when the path uses a mix of nodes that either belong to V or not. In such a case all these paths must end with some node $v' \in V - S_u(V)$. The path from v' will utilize the wormhole link and will try to close the loop using some k-hop neighbor u' of u. There are two cases again to consider. Either $v' \in S_{u'}(V)$ or not.

In the first case, u' overheard a packet containing the ID of v' and placed v' in a quarantine (suspect list). But then it cannot have moved v' in its neighborhood list unless it had performed a "small path" test with v'. Thus u will never be fooled in accepting v' as a legitimate neighbor of u' since it always has up-to-date information regarding the neighborhood structure of u'. In the second case, u' never heard anything about v' so it definitely does not have v' in its neighbors list. Thus again u will reject the path. In summary, no wormhole can be established between the sets U and V.

Case 2: Consider now the benign case where no attack takes place (V may be a set of newly added nodes). Node u will attempt to establish a small path with v, excluding the suspect nodes. This can happen in a straightforward way and we leave the detailed description for Section 4.3.

We need to point out, however, that the algorithm may fail to find short paths[2] and as a result treat these nodes as part of a wormhole attack. This is an instance of the "safe failure" principle we highlighted in the beginning of this section. *We opted for preventing a wormhole link from being established when an attack is taking place at the expense of not allowing some legitimate nodes to be part of the network when no attack occurs.* However, as we will demonstrate probabilistically in the next section and verify experimentally in Section 5, the probability of this happening is very small even for low density networks.

4.2 Existence of Short Paths - Probabilistic Analysis

We will now argue about the existence of small paths between two nodes u and v. We model the topology of the wireless sensor network by a random geometric graph which can be constructed as follows: we throw n nodes uniformly at random onto the surface of a unit square and we connect all nodes within distance r of each other. An edge (u_i, u_j) in the geometric graph corresponds to a communication link between sensors i and j. The analysis will be performed in terms of the following quantities (that are either given or easily derived):

- the number of sensors in the field, n;
- the communication range, r;
- the probability p that two random nodes u and v are neighbors;
- the average density d of the graph, i.e. the expected number of neighbors of a given node.

[2] Either because of lack of common neighbors with v or because some neighborhood updates were lost.

In what follows, we will express all results in terms of d. If u is a node with range r in the unit square, the probability p that another node v is a neighbor of u is given by the quantity $p = \pi r^2$. Thus the expected number of neighbors d of u is equal to $d = (n-1)p \approx np = n\pi r^2$, for large enough n.

Our goal is to upper bound the probability that no path exists between u and v when 2-hop neighborhoods are known. It can be shown (the proof is omitted due to space restrictions) that

$$\Pr[\text{No small path between } u \text{ and } v] < e^{-0.58d},$$

where d is the average network density. Thus the probability that u and v will be connected is lower bounded by $1 - e^{-0.58d}$. This probability is very high even for low density networks (more than 94% when $d = 5$ and more than 99% when $d = 10$). And as we will see in Section 5, this probability increases even further when we consider paths between 1-hop neighbors of u and 1-hop neighbors of v.

4.3 Algorithm Description

We will now present in more details the Wormhole Detection Algorithm (WDA) described in the previous sections. Recall that WDA is to look for a simple path between two nodes u and v that indicates absence of a wormhole link between them. The algorithm is strictly *localized* and therefore only nodes affected by a change in the neighborhood topology (i.e., "hearing" of a new node) need to run it. Each node u, upon discovery of a new *suspect* node v, searches for such paths using its k-hop neighborhood knowledge. While the algorithm may run for general k-hop detection, our simulation studies (presented in Section 5) showed that $k \leq 2$ is sufficient for various densities of a network.

Algorithm 1. The *Wormhole Detection* Algorithm

```
Data: $SuspectList(u)$ and $k$-hop Neighborhood Information of $u$ where $k = 1, 2$.
Result: For each node $v_i$ in $SuspectList(u)$, compute whether $v_i$ is legitimate
        or not.
begin
    for every $v_i$ in $SuspectList(u)$ do
        $v_i \twoheadrightarrow$ NOT legitimate;
        /** Look for small path between $[u, v_i]$ */
        if $[ImmediatePath(u, v_i)] \vee [1\text{-}HopPath(u, v_i)] \vee [2\text{-}HopPath(u, v_i)]$
        then
            /** A path has been found */
            $v_i \twoheadrightarrow$ legitimate;
        end
        continue with next node $v_i$ in $SuspectList(u)$;
    end
end
```

The code of WDA is given in Algorithm 1. We will refer to the set of *suspected nodes* of each node u, as $SuspectList(u)$. A node v is labeled as suspect when it is the first time we hear from it and we want to perform the local path existence test described in Section 4.1. As we mentioned above, the output of this test indicates whether this suspect node is *legitimate* (and can be safely added to the neighborhood list of u) or might be the result of a wormhole in progress.

Each node u maintains the list of 2-hop neighbors $N^2(u)$. WDA performs the path existence test for all nodes $v \in SuspectList(u)$. The test consists of three steps, each of which is activated upon *failure* of the previous one.

$ImmediatePath(u, v)$: This step tries to identify a *direct path* between the testing pair $[u, v]$. It works by having the active node u check if at least one of v's neighbors is included in its neighborhood list. This is possible since u knows the 1-hop neighbors of v. In order to perform this check, u traverses both it's own and v's neighborhood lists. Thus, the time needed is $O(d^2)$, where d is the average density of the network.

1-$HopPath(u, v)$: Active node u checks whether one of its 1-hop neighbors is *directly* connected to one of the 1-hop neighbors of v. This requires node u to traverse the neighborhood lists of all its 1-hop neighbors and check if at least one of v's neighbors is included. This takes time $O(d^3)$, where d is the average density of the network. Intuitively, this is done by considering whether u and v have two neighbors x and y, respectively, that are *directly* connected.

2-$HopPath(u, v)$: In this last step node u checks whether one of its 1-hop neighbors shares a common neighbor with one of the 1-hop neighbors of v. Thus, the time needed is of order $O(d^4)$. This is done by considering whether u and v have two neighbors x and y, respectively, that have a common 1-hop neighbor z. This alternative path, if it exists, will be an attestation that v is a legitimate node. Otherwise, node u can conclude with very high probability that "hearing" v is the result of a wormhole, without the need of checking the existence of k-hop paths for $k > 2$.

5 Simulation Results

In this section, we present the practical impacts of the proposed wormhole detection algorithm. The experiments were deployed both in a simulator and a real sensor environment. Two properties were of special interest: *i*) Path Existence Percentage, and *ii*) Detection Time. Since the algorithm always prevents a wormhole attack, our focus is mostly on legitimate node addition.

5.1 Performance Evaluation

In order to evaluate our wormhole detection algorithm, we tested its success in finding small paths between a pair of nodes $[u, v]$. We generated random network topologies by placing 500 nodes uniformly at random with an average density d varying between 4 and 15. To ensure statistical validity, we repeated each experiment 1000 times and averaged the results. Once a topology was created, we started randomly adding new nodes in the network. This triggered the

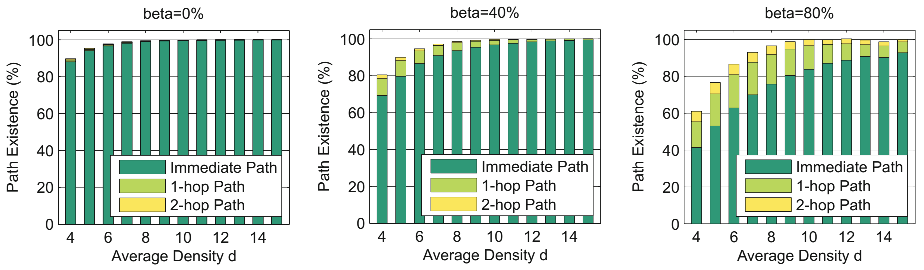


Fig. 3. Path Existence percentage between a pair of nodes $[u, v]$ when a fraction, *beta*, of u's neighbors are excluded from the WD algorithm. (a) beta = 0% (b) beta = 40% ((c) beta = 80%.

surrounding nodes to run WDA. In order to achieve real scenario simulations, we had to take into consideration possible missed route updates or incomplete neighborhood information. Thus, we used a variable -*beta*- which indicated the percentage of a node's neighbors that were excluded during the path existence test. In other words, *beta* is a fraction of u's neighbors that are not taken into account when u is trying to find a *small* path with v. These "excluded nodes" were selected at random from u's neighborhood, for increasing values of *beta*.

Figure 3 depicts the path existence percentage between a pair of nodes $[u, v]$. This was broken down to the existence of an *immediate* path (1^{st} step of WDA), *1-hop* path (2^{nd} step), and *2-hop* path (3^{rd} step). As we can see, our algorithm provides very good results even for low densities and when a large number of u's neighbors is excluded. In general, the following observations can be made by the simulation results:

– WDA provides very good results (almost 100% success in finding a small path) despite the network density (Figure 3(a)).
– The more expensive 1-hop and 2-hop path tests are rarely need to be executed since in most of the cases the immediate path test is sufficient. This results in faster detection. Also, it loosens the need of the 2-hop neighborhood maintenance by a node (Assumption SMA-1).
– Existence probability does drop for large *beta*, but only for low density cases. However, in such cases, the usefulness of the network also drops. Even when *beta* = 80% (Figure 3(c)) the existence percentage is almost 100% for densities $d \geq 7$. Thus, one may reside only in the 1-hop test to increase the success probability. The usefulness of the 2-hop test seems to be questionable since even for large *beta* (80%) its contribution remains small. This suggests that the 2-hop test can be dropped entirely.

In summary, our algorithm allows legitimate nodes to be added with high probability even for low density networks. In such networks, one can adjust WDA to just consider 1-hop neighborhoods and perform only the first check (for more, see also Section 6). Furthermore, since the algorithm does not progress to the next test unless the previous one has failed, in more than 95% of the cases WDA will conclude after the first check, keeping the overall test time relatively small.

5.2 Implementation and Experiments on Real Sensor Devices

In this section, we present some results on the detection time of the algorithm. These are collected through experiments from our implementation of WDA on real sensor devices[3]. The goal is to justify the practicality of our approach from an implementation and real deployment point of view.

Table 1. Size of the compiled code, in bytes

Module	RAM usage	Code Size
Neighborhood Discovery	136	968
WDA	104	766
Total	**240**	**1734**

We start with the *memory footprint*, an important measure of WDA's feasibility and usefulness on memory constrained sensor nodes. Table 1 lists the memory requirements of the necessary modules. The Neighborhood Discovery module is the one responsible for creating a node's k-hop neighborhood ($k = 1, 2$). The WDA module contains all the necessary methods described in Section 4.3. As we can see, in total, the algorithm consumes 240 bytes of RAM and 1734 bytes of code memory. This leaves enough space in the mote's memory for user applications and other protocols. For example, the total RAM available in Telos motes is 10 KB.

Next we measured the time each test of WDA required. Figure 4(a) shows the measured mean times for each of the three tests, for different network densities. The immediate and 1-hop path tests, which cover more than 95% of the cases, conclude in much less than 1 second (around 100ms and 550ms, respectively). The most time consuming step is the 2-hop test. However, as we mentioned before, it will rarely need to be executed and it can even be dropped entirely.

We have to note that this experiment was conducted with nodes maintaining their neighbors in a typical list that is not sorted or pre-processed in any way. However, as Figure 4(b) shows, having neighbor lists sorted by node IDs significantly decreases the detection time of WDA's steps, and in particular that of the 2-hop test. In any case, the time of WDA is very small fulfilling the need of immediate response in case of a wormhole in progress. Dropping the 2-hop test entirely does not significantly affect the success probability while at the same time it decreases the running time considerably. In such case there is no need to resort in sorted neighborhoods or other data structures to speedup computation, however, we leave this decision open to the particular implementation.

[3] The current development of WDA builds on Moteiv Telos motes - a popular architecture in the sensor network research community.

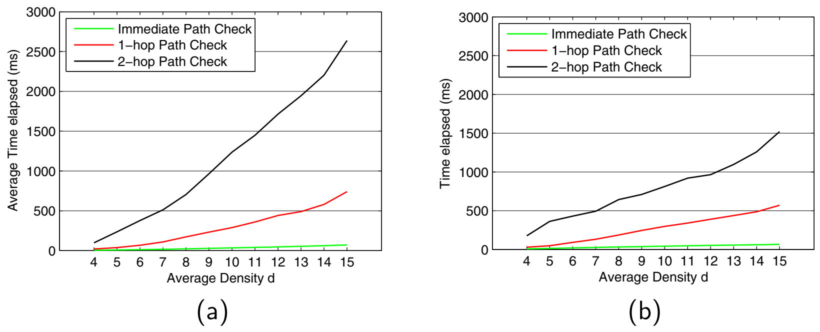


Fig. 4. Running time of WDA for different network densities. (a) Node IDs are randomly placed in a neighborhood list (b) Neighborhood list is kept sorted.

6 Discussion and Critique

In this work we proposed a method that identifies and prevents wormhole attacks. Key to the method is the observation that in the case of an attack, no real path of just a few hops will ever exist between the wormhole endpoints (Section 3.2, Assumption AMA-1) since otherwise it would not be possible to distinguish between real and fake short paths. It is, however, a very realistic assumption since the point of the attack is to make a distant node appear closer to a point of interest (say the base station) and attract as much traffic as possible [1,13]. Decreasing the radius of the wormhole, will also decrease the effect of the attack on the network in terms of the numbers of sensors that use the wormhole link [2].

Of course an attacker can still try to fool the algorithm by establishing many smaller wormholes that are connected in a series but this has two shortcomings. First, it takes a lot of time, effort and hardware which will increase the risks of detection. Second, it can be defeated easily by our method if we just consider 1-hop neighborhoods ($k = 1$). In this case no wormhole link of length bigger than 2 can ever be established while, as demonstrated in Figure 3, addition of legitimate nodes can occur with high probability even for small density networks.

For the same reason, assumption SMA-1 may also be weakened by maintaining only 1-hop neighborhoods among nodes, a typical procedure for any routing protocol. In practice well connected networks have node densities bigger than 7 or 8 in which case the overhead of maintaining 2-hop neighborhoods does not offer any significant advantage over the 1-hop case (Figure 3). Our only important assumption is that there is some initial interval where the nodes have safely established their neighborhood information (Assumption AMA-2). This is a standard assumption made in many of the works in this area [2,13,14,17,18,19] but also in works where a security infrastructure has to be established.

We should mention that one way our method could be defeated is if the attacker has compromised a few nodes in the neighborhood of the wormhole endpoints in which case the compromised nodes may "lie" about their true neighbors. This combined attack, however, is *not* a wormhole attack anymore.

All cited results consider the attack in which an adversary *simply* forwards messages from one part of the network to another. This combined attack, however, is an interesting future direction that we are planning to pursue further.

Finally, we should emphasize that contrary to prior work [2,14], we don't assume that the network is *static*. In fact a large portion of this work considers *dynamic* changes in the neighborhood structure of nodes.

7 Conclusions

A wormhole attack is considered to be a prominent attack that is carried out in order to alter the correct functioning of a wireless network. The detection of such an attack is still a significantly challenging task. In this paper, we have studied the problem of wormhole detection in sensor networks. We have provided a snapshot of the current state of the art, discussed existing solutions and listed their behavior and limitations. More importantly, however, we have proposed a practical detection algorithm based on the observation that no real path of just a few hops will ever exist between the wormhole endpoints.

We investigated the effectiveness of our proposed algorithm both analytically and experimentally. Our results have confirmed that it can always prevent a wormhole attack, while at the same time it allows legitimate node addition with high probability, even for low density networks. Furthermore, the implementation details show that it is lightweight enough to run on sensor nodes in terms of both memory requirements and processing overhead. In general, we believe that this algorithm will have a practical use in real-world deployments and can be considered as a reference point for further investigation of more attractive solutions against wormhole attacks since it does not require any specialized hardware, tight clock synchronization, or distance measurements between the nodes.

Acknowledgments. This work has been funded by the European Community's FP7 program, Call ref. SEC-2007-1, under Grant Agreement no: 217925.

References

1. Ahmed, N., Kanhere, S.S., Jha, S.: The holes problem in wireless sensor networks: a survey. Mobile Computing and Communications Review 9(2), 4–18 (2005)
2. Buttyán, L., Dóra, L., Vajda, I.: Statistical wormhole detection in sensor networks, pp. 128–141 (2005)
3. Eriksson, J., Krishnamurthy, S.V., Faloutsos, M.: Truelink: A practical countermeasure to the wormhole attack in wireless networks, pp. 75–84 (2006)
4. Hai, T.H., Huh, E.-N.: Detecting selective forwarding attacks in wireless sensor networks using two-hops neighbor knowledge. In: NCA 2008: Proceedings of the 2008 Seventh IEEE International Symposium on Network Computing and Applications, Washington, DC, USA, pp. 325–331. IEEE Computer Society, Los Alamitos (2008)
5. Hu, Y.-C., Perrig, A., Johnson, D.B.: Wormhole attacks in wireless networks. IEEE Journal on Selected Areas in Communications 24(2), 370–380 (2006)

6. Issa Khalil, N.B.S., Bagchi, S.: Liteworp: A lightweight countermeasure for the wormhole attack in multihop wireless networks, p. 22 (June 2005)
7. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: Attacks and countermeasures. AdHoc Networks Journal 1(2-3), 293–315 (2003)
8. Korkmaz, T.: Verifying physical presence of neighbors against replay-based attacks in wireless ad hoc networks. In: ITCC 2005: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), Washington, DC, USA, vol. II, pp. 704–709. IEEE Computer Society, Los Alamitos (2005)
9. Krontiris, I., Benenson, Z., Giannetsos, T., Freiling, F., Dimitriou, T.: Cooperative intrusion detection in wsn. In: Roedig, U., Sreenan, C.J. (eds.) EWSN 2009. LNCS, vol. 5432, pp. 263–278. Springer, Heidelberg (2009)
10. Krontiris, I., Giannetsos, T., Dimitriou, T.: Launching a sinkhole attack in wireless sensor networks; the intruder side. In: SecPriWiMob 2008: First International Workshop on Security and Privacy in Wireless and Mobile Computing, Networking and Communications, Avignon, France, October 12-14 (2008)
11. Lehsaini, M., Guyennet, H., Feham, M.: A-coverage scheme for wireless sensor networks. In: ICWMC 2008: Proceedings of the 2008 The Fourth International Conference on Wireless and Mobile Communications, Washington, DC, USA, pp. 91–96. IEEE Computer Society, Los Alamitos (2008)
12. Lehsaini, M., Guyennet, H., Feham, M.: CES: Cluster-based energy-efficient scheme for mobile wireless sensor networks. In: IFIP Conference on Wireless Sensor and Actor Networks, Ontario, Canada (July 2008)
13. Maheshwari, R., Gao, J., Das, S.R.: Detecting wormhole attacks in wireless networks using connectivity information. In: 26th IEEE International Conference on Computer Communications, INFOCOM 2007, pp. 107–115. IEEE, Los Alamitos (2007)
14. Poovendran, R., Lazos, L.: A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks. Wireless Networks 13(1), 27–59 (2007)
15. Capkun, S.S., Buttyan, L., Hubaux, J.-P.: Sector: secure tracking of node encounters in multi-hop wireless networks. In: SASN 2003: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks, pp. 21–32. ACM Press, New York (2003)
16. Savvides, A., Han, C.-C., Strivastava, M.B.: Dynamic fine-grained localization in ad-hoc networks of sensors. In: MobiCom 2001: Proceedings of the 7th annual international conference on Mobile computing and networking, pp. 166–179. ACM Press, New York (2001)
17. Bhargava, B., Wang, W.: Visualization of wormholes in sensor networks. In: Proceedings of ACM Workshop on Wireless Security (WiSe), in conjunction with MobiCom (October 2004)
18. Wang, W., Bhargava, B., Lu, Y., Wu, X.: Defending against wormhole attacks in mobile ad hoc networks: Research articles. Wirel. Commun. Mob. Comput. 6(4), 483–503 (2006)
19. Wang, W., Lu, A.: Interactive wormhole detection and evaluation. Information Visualization 6(1), 3–17 (2007)

Wireless Jamming Localization by Exploiting Nodes' Hearing Ranges

Zhenhua Liu[1], Hongbo Liu[2], Wenyuan Xu[1], and Yingying Chen[2,⋆]

[1] Dept. of CSE, University of South Carolina
{liuz,wyxu}@cse.sc.edu
[2] Dept. of ECE, Stevens Institute of Technology
{hliu3,yingying.chen}@stevens.edu

Abstract. Jamming attacks are especially harmful when ensuring the dependability of wireless communication. Finding the position of a jammer will enable the network to actively exploit a wide range of defense strategies. Thus, in this paper, we focus on developing mechanisms to localize a jammer. We first conduct jamming effect analysis to examine how a hearing range, e.g., the area from which a node can successfully receive and decode the packet, alters with the jammer's location and transmission power. Then, we show that the affected hearing range can be estimated purely by examining the network topology changes caused by jamming attacks. As such, we solve the jammer location estimation by constructing a least-squares problem, which exploits the changes of the hearing ranges. Compared with our previous iterative-search-based virtual force algorithm, our proposed hearing-range-based algorithm exhibits lower computational cost (i.e., one-step instead of iterative searches) and higher localization accuracy.

1 Introduction

The rapid advancement of wireless technologies has enabled a broad class of new applications utilizing wireless networks, such as patient tracking and monitoring via sensors, traffic monitoring through vehicular ad hoc networks, and emergency rescue and recovery based on the availability of wireless signals. To ensure the successful deployment of these pervasive applications, the dependability of the underneath wireless communication becomes utmost important. One threat that is especially harmful is jamming attacks. The broadcast-based communication combined with the increasingly flexible programming interference of commodity devices makes launching jamming attacks with little effort. For instance, an adversary can easily purchase a commodity device and reprogram it to introduce packet collisions that force repeated backoff of other legitimate users and thus, disrupt network communications.

⋆ The work was supported by National Science Foundation Grants CNS-0845671 and CNS-0954020.

To ensure the dependability of wireless communication, much work has been done to detect and defend against jamming attacks. In terms of detection, single-statistics-based [13] and consistent-check-based algorithms [15] have been proposed. The existing countermeasures for coping with jamming include two types: the proactive conventional physical-layer techniques that provide resilience to interference by employing advanced transceivers [10], e.g., frequency hopping, and the reactive non-physical-layer strategies that defend against jamming leveraging MAC or network layer mechanisms, e.g., adaptive error correcting codes [6], channel adaption [14], spatial relocation [5], or constructing wormholes [2].

Few studies have been done in identifying the physical location of a jammer. However, localizing a jammer is an important task, which not only allows the network to actively exploit a wide range of defense strategies but also provides important information for network operations in various layers. For instance, a routing protocol can choose a route that does not traverse the jammed region to avoid wasting resources caused by failed packet deliveries. Alternatively, once a jammer's location is identified, one can eliminate the jammer from the network by neutralizing it. In light of the benefits, in this paper, we address the problem of localizing a jammer.

Although there have been active research in the area of localizing a wireless device [12,1,3], most of those localization schemes are inapplicable to jamming scenarios. For instance, many localization schemes require the wireless device to be equipped with specialized hardware [9,12], e.g., ultrasound or infrared, or utilize signals transmitted from wireless devices to perform localization. Unfortunately, the jammer will not cooperate and the jamming signal is usually embedded in the legal signal and thus, is hard to extract, making the signal-based and special-hardware-based approaches inapplicable.

Recent work [4,7] on jamming localization algorithms is iterative-search-based. Without presenting performance evaluation, Pelechrinis *et al.* [7] proposed to localize the jamming by measuring packet delivery rate (PDR) and performing gradient descent search. Liu *et al.* [4] utilized the network topology changes caused by jamming attacks and estimated the jammer's position by introducing the concept of virtual forces. The virtual forces are derived from the node states and can guide the estimated location of the jammer towards its true position iteratively.

In this paper, we proposed a hearing-range-based localization scheme that also exploits the network topology changes caused by jamming attacks. In particular, to quantify the network topology changes, we introduced the concept of a node's hearing range, an area from which a node can successfully receive and decode the packet. We have discovered that a jammer may reduce the size of a node's hearing range, and the level of changes is determined by the relative location of the jammer and its jamming intensity. Therefore, instead of searching for the jammer's position iteratively, we can utilize the hearing range to localize the jammer in one round, which significantly reduces the computational cost yet achieves better localization performance than prior work [4].

We organize the remainder of the paper as follows: we specify our jamming attack model and provide an analysis on jamming effects by introducing the concept of the hearing range in Section 2. Then, we present our hearing-range-based algorithm in Section 3. In Section 4, we conduct simulation evaluation and present the performance results. Finally, we conclude in Section 5.

2 Analysis of Jamming Effects

In this section, we start by outlining the basic wireless network that we use throughout this paper and briefly reviewing the theoretical underpinning for analyzing the jamming effects. Then, we study the impact of a jammer on the wireless communication at two levels: the individual communication range level and the network topology level.

2.1 Network Model and Assumptions

We target to design our solutions for a category of wireless networks with the following characteristics.

Neighbor-Aware and Location-Aware. Each node in the network maintains a table that stores its neighbor information. Each node is aware of its own location and its neighbors' locations. This is a reasonable assumption as many applications require localization services [1]. Each node is able to sense the changes on its neighbor table by comparing the current neighbor table with the previous one. Further, we assume that the node is stationary and transmits the signal at the same transmission power level by using an omnidirectional antenna. Mobility will be considered in our future work.

Adaptive-CCA. Clear channel assessment (CCA) is an essential component of Carrier Sense Multiple Access (CSMA), the de-facto medium access control (MAC) protocols in many wireless networks. In particular, each network node is only allowed to transmit packets when the channel is idle by using CCA as channel detection. Typically, CCA involves having wireless devices monitoring the received signal and comparing the average received signal strength with a threshold Υ. Studies [8] have shown that adaptive-CCA, which adjusts the threshold Υ based on the ambient noise floor, can achieve better throughput and latency than using a pre-determined threshold Υ. Therefore, we assume that each node employs an adaptive-CCA mechanism in our study.

In this work, we focus on locating a jammer after it is detected. Thus, we assume the network is able to identify a jamming attack, leveraging the existing jamming detection approaches [13,15].

2.2 Communication in Non-jamming Scenarios

Before analyzing the impact of jamming on the communication range, we briefly review the key factors that affect packet deliveries. Essentially, the MAC layer concept, packet delivery ratio (PDR), is determined by the physical metric, signal-to-noise ratio (SNR). At the bit level, the bit error rate (BER) depends on the probability that a receiver can detect and process the signal correctly.

To process a signal and derive the associated bit information with high probability, the signal has to exceed the noise by certain amount. Given the same hardware design of wireless devices, the minimum required surplus of signals over ambient noise is roughly the same. We use γ_o to denote the *minimum SNR*, the threshold value required to decode a signal successfully. We consider that a node A is unable to receive messages from node B when $(SNR)_{B\rightarrow A} < \gamma_o$, where $(SNR)_{B\rightarrow A}$ denotes the SNR of messages sent by B measured at A.

The communication range defines a node's ability to communicate with others, and it can be divided into two components: the ***hearing range*** and the ***sending range***. Consider node A as a receiver, the hearing range of A specifies the area within which the potential transmitters can deliver their messages to A, e.g. for any transmitter S in A's hearing range, we have $(SNR)_{S\rightarrow A} > \gamma_o$. Similarly, consider A as a transmitter, the sending range of A defines the region within which the potential receivers have to be located to receive messages sent by A, e.g., for any receiver R in A's sending range, we have $(SNR)_{A\rightarrow R} > \gamma_o$.

Consider the standard free-space propagation model, the received power is

$$P_R = \frac{P_T G}{4\pi d^2}, \tag{1}$$

where P_T is the transmission power, G is the product of the sending and receiving antenna gain in the LOS (line-of-sight) between the receiver and the transmitter, and d is the distance between them.

In a non-jamming scenario, the average ambient noise floor, P_N, across the entire space will be the same. Since the received signal power is a function of d^2, both the hearing range and the sending range of node A will be the same, a circle centered at A with a radius of $r_c = \sqrt{\frac{P_T G}{4\pi\gamma_o P_N}}$. This observation coincides with the common knowledge, that is, the communication between a pair of nodes is bidirectional when there are no interference sources.

2.3 The Effect of Jamming on the Communication Range

Applying the free-space model to a jammer, the jamming signals also attenuate with distance, and they reduce to the normal ambient noise level at a circle centered at the jammer. We call this circle the *Noise Level Boundary (NLB)* of the jammer. Since jamming signals are nothing but interference signals that contribute to the noise, a node located within the NLB circle will have bigger ambient noise floor than the one prior to jamming.

For simplicity, much work assumes that when a node is located inside the jammer's NLB circle it loses its communication ability completely, e.g., both its sending range and hearing range become zero. Such assumptions may be valid for nodes that perform CCA by comparing the channel energy with a fixed threshold, as all nodes within the NLB will consider the channel busy throughout the duration that the jammer is active. However, in a network where adaptive-CCA is used, the nodes inside the jamming's NLB circle will still maintain partial communication ability yet weaker than the nodes outside the NLB circle.

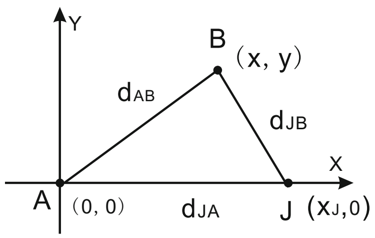


Fig. 1. The coordinate system for the hearing range and the sending range of node A, wherein A and B are legitimate nodes, and J is the jammer

In this paper, we will focus on examining the hearing range changes caused by jamming, and we refer readers to our prior work on the analysis of the sending range [16]. In particular, we consider a simple network consisting of three players: a jammer J interferes with the legitimate communications from the transmitter B to the receiver A, as depicted in Figure 1. The legitimate node transmits at the power level of P_T, and the jammer interferes at the power level of at P_J.

The signal-to-noise ratio at A when the jammer J is active is $(SNR)_{B\to A} = P_{BA}/(P_N + P_{JA})$, where P_{BA} and P_{JA} are the received power of B's signal and the jamming signal at node A, respectively. Assume that the jammer uses the same type of devices as the network nodes, e.g., both use omnidirectional antennas, then the antenna gain product between J and A, and the one between B and A are the same.

Let's first examine the cases when node A observes a jamming signal much larger than the normal ambient noise P_N, then

$$(SNR)_{B\to A} \approx \frac{P_T d_{JA}^2}{P_J d_{AB}^2}. \tag{2}$$

To find the new hearing range under jamming attacks, we search for locations $B = (x, y)$ that satisfy the equations: $(SNR)_{B\to A} = \gamma_o$. Substituting $d_{AB}^2 = x^2 + y^2$ and $d_{JA}^2 = x_j^2$ to Equation 2, node A's hearing range when the jamming signal is dominant can be expressed as

$$x^2 + y^2 = \frac{x_j^2}{\beta}, \tag{3}$$

where $\beta = \frac{\gamma_o}{P_T/P_J}$. Thus, the hearing range of node A is a circle centered at itself with a radius of $r_h = \frac{|x_j|}{\sqrt{\beta}}$. This formula coincides with the intuition: for the same x_j, a louder jamming signal affects legitimate nodes more; given P_T and P_J, the closer a legitimate node is located to the jammer, the smaller its hearing range becomes, as illustrated in Figure 2.

Now let's turn to the cases where the jamming signal no longer dominates the ambient noise, e.g., when nodes are located close to the edge of jammer's NLB, as illustrated in Figure 2 (b). The hearing range becomes:

$$x^2 + y^2 = \frac{x_j^2 P_T}{\gamma_o (x_j^2 \eta + P_J)}, \tag{4}$$

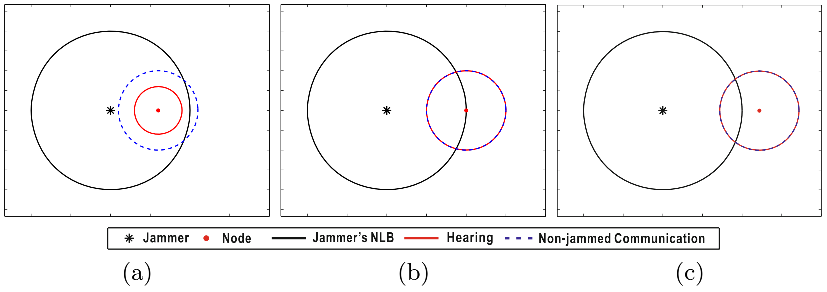


Fig. 2. The hearing range and non-jammed communication range when the location of a jammer is fixed and a node is placed at different spots: (a) inside the jammer's NLB; (b) at the edge of the jammer's NLB; (c) outside the jammer's NLB

where $\eta = 4\pi P_N/G$. To avoid the complexities of deriving the antenna gain G, we approximate the node's hearing range with its normal hearing range, e.g., the hearing range without jammers. In summary, the hearing range of a node A is a circle centered at A with a radius of $r_h = \min\left(\frac{|x_j|}{\sqrt{\beta}}, \sqrt{\frac{P_T G}{4\pi\gamma_o P_N}}\right)$, as illustrated in Figure 2.

2.4 The Effect of Jamming on Network Topology

In this section, we extend our analysis of jamming impact from the individual node level to the network level, and classify the network nodes based on the level of disturbance caused by the jammer.

Essentially, the communication range changes caused by jamming are reflected by the changes of neighbors at the network topology level. We note that both the hearing range and the sending range shrink due to jamming. We choose to utilize the change of the hearing range, since it is easier to estimate, e.g., estimation only involves receiving at each node. We define that node B is a neighbor of node A if A can *receive* messages from B. Based on the degree of neighbor changes, we divide the network nodes under jamming attacks into the following three categories:

– **Unaffected Node.** The unaffected node may have a slightly changed hearing range, but its neighbor list remains unchanged, e.g., it can still hear from all its original neighbors. We note that the unaffected node does not have to be outside the jammer's NLB.
– **Boundary Node.** The hearing range of a boundary node is reduced, and the number of nodes in its neighbor list is also decreased. More importantly, it can still receive information from all unaffected nodes within finite steps.
– **Jammed Node.** The hearing range of a jammed node has been severely disturbed. We define a jammed node as the one that does not have any unaffected nodes or boundary nodes in its neighbor list, i.e., no unaffected nodes or boundary nodes within its hearing range. We note that it is possible that a few jammed nodes can hear each other and form a "Jammed Cluster".

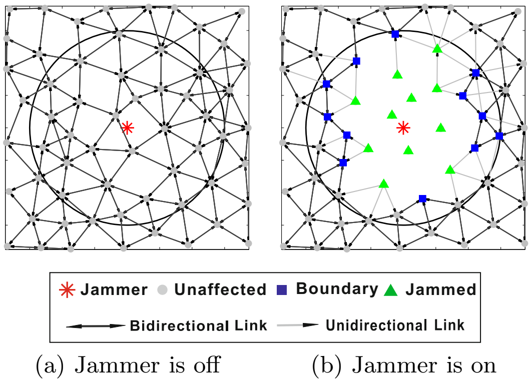


Fig. 3. An example of the topology change of a wireless network due to jamming, where the black solid circle represents the jammer's NLB

However, they are isolated and cannot receive information from the majority of the networks.

Figure 3 illustrates an example of network topology changes caused by a jammer. Prior to jamming, neighboring nodes were connected through bidirectional links. Once the jammer became active, nodes lost their bidirectional links either partially or completely. In particular, the nodes marked as triangles lost all their inbound links (receiving links) from their neighbors and became jammed nodes. Interestingly, some jammed nodes can still send messages to their neighbors, and they may participate in the jamming localization by delivering information to unaffected nodes as described in Section 3. The nodes depicted in rectangles are boundary nodes. They lost part of its neighbors but still maintained partial receiving links, e.g., at least connected to one unaffected nodes either directly or indirectly. Finally, the rest of nodes are unaffected nodes and they can still receive from all their neighbors.

3 Jammer Localization Algorithm

3.1 Algorithm Description

In the previous sections, we have shown that the hearing range of a node may shrink when a jammer becomes active, and the level of change is determined by the distance to the jammer and the strength of the jamming signals. As the example illustrated in Figure 1, if B happens to be located at the edge of A's hearing range, then we have $(SNR)_{B\to A} \approx \gamma_o$ and $d_{AB} = r_{h_A}$. Therefore, we can convert Equation 2 into a general form,

$$(x_A - x_J)^2 + (y_A - y_J)^2 = \beta r_{h_A}^2, \tag{5}$$

where r_{h_A} is the new hearing range of node A, $\beta = \frac{\gamma_o}{P_T/P_J}$, and (x_A,y_A) and (x_J,y_J) are the coordinates of node A and the jammer J, respectively.

Suppose that due to jamming the hearing ranges of m nodes have shrunk to r_{h_i}, $i = \{1, \ldots, m\}$. Then, we have m equations:

$$\begin{aligned}(x_1 - x_J)^2 + (y_1 - y_J)^2 &= \beta r_{h_1}^2 \\ (x_2 - x_J)^2 + (y_2 - y_J)^2 &= \beta r_{h_2}^2 \\ &\vdots \\ (x_m - x_J)^2 + (y_m - y_J)^2 &= \beta r_{h_m}^2\end{aligned} \tag{6}$$

Assume that we can obtain r_{h_i} for each of m nodes, then we can localize the jammer by solving the above equations. To avoid solving a complicated nonlinear equations, we first linearize the problem by subtracting the m^{th} equation from both sides of the first $m-1$ equations and obtain linear equations in the form of $\mathbf{Az} = \mathbf{b}$ with

$$\mathbf{A} = \begin{pmatrix} x_1 - x_m & y_1 - y_m & \frac{1}{2}(r_{h_1}^2 - r_{h_m}^2) \\ \vdots & \vdots & \vdots \\ x_{m-1} - x_m & y_{m-1} - y_m & \frac{1}{2}(r_{h_{m-1}}^2 - r_{h_m}^2) \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} (x_1^2 - x_m^2) + (y_1^2 - y_m^2) \\ \vdots \\ (x_{m-1}^2 - x_m^2) + (y_{m-1}^2 - y_m^2) \end{pmatrix}.$$

The least squares solution can be calculated by

$$\mathbf{z} = [x_J, y_J, \beta]^T = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{7}$$

3.2 Algorithm Challenges

To localize a jammer using the aforementioned solution, two questions have to be answered: (1) how to estimate the radius of a node's hearing range (aka. the hearing radius), and (2) what is the criteria of selecting nodes as candidates to form the equation group?

Estimating the Hearing Radius. The basic idea of estimating the hearing radius of node A is to identify the furthest neighbor that A can hear from and the closest node that A *cannot* hear. Since the distances to those two special nodes provide the lower bound and the upper bound of A's hearing radius, we can estimate A's hearing radius as the mean value of those bounds. Applying this observation to a jamming scenario, a node can leverage the change of its neighbor list to identify those two specially-located nodes. Figure 4 illustrates such an example.

Before the jammer started to disturb the network communication, node A had a neighbor list of $\{n_1, n_2, n_3, n_4, n_5\}$. Once the jammer became active, A's neighbors reduced to $\{n_2, n_4\}$ and we call this set the *Remaining Neighbor Set.*

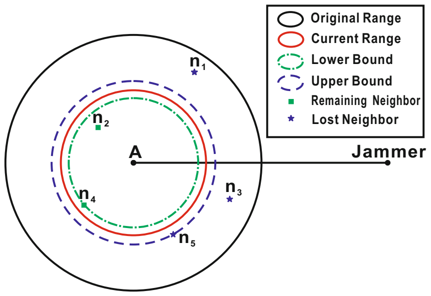


Fig. 4. An illustration of estimating the hearing range of node A leveraging the change of its neighbor list

At the same time, A can no longer hear from $\{n_1, n_3, n_5\}$, the *Lost Neighbor Set.* The estimated upper bound of A's hearing radius, r_u, equals the distance to n_5, the nearest node in the lost neighbor set; the estimated lower bound, r_l, equals the distance to n_4, the furthest node in the remaining neighbor set. As a result, the true hearing radius r_{h_A} is sandwiched between $[r_l, r_u]$ and can be estimated as $\hat{r}_{h_A} = (r_u + r_l)/2$.

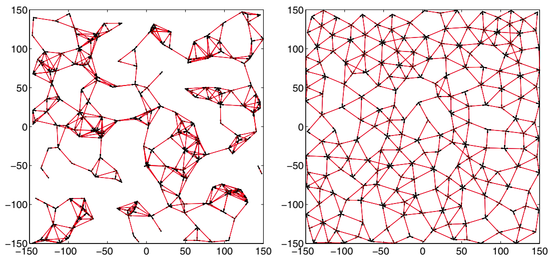
The estimation error of the hearing radius, e_h, depends on $(r_u - r_l)$ and can be any value in $[0, (r_u - r_l)/2]$. When the distances between any two nodes are uniformly distributed, the estimation error e_h follows uniform distribution with the expected value as $\frac{r_u - r_l}{4}$.

Selecting *m* Nodes. The nodes that can contribute to the jamming localization have to satisfy the following requirements: (1) they have a reduced hearing range; (2) the new hearing range under jamming attacks can be estimated; and (3) they are able to transmit their new hearing radius out of the jammed area.

Although an unaffected node may have a slightly reduced hearing range, its neighbor list remains unchanged. Therefore, its hearing radius cannot be estimated and neither can it contribute an equation to localize the jammer. Likewise, although a jammed node's hearing range is decreased severely, its remaining neighbor set may be empty, preventing it from estimating the up-to-date hearing radius accurately. Even in cases when they may estimate their hearing ranges with the help of "Jammed Cluster", they may not be able to transmit their estimations out of the jammed area due to communication isolation. In short, most of the jammed nodes are not suitable for jamming localization. Only those that have more than one neighbor and are able to send out messages to unaffected nodes can be used.

Finally, with regard to boundary nodes, the hearing range of a boundary node is reduced. Leveraging their reduced neighbor lists, their hearing radii can be estimated. More importantly, they can still communicate with unaffected nodes within finite steps. Therefore, all boundary nodes shall be used to participate the jamming localization.

In summary, we use all the boundary nodes and some jammed nodes to form the equation group for jamming localization.



(a) Simple Deployment (b) Smart Deployment

Fig. 5. Two deployments on a network with 200 nodes

4 Experiment Validation

4.1 Experiment Setup and Performance Metrics

We now evaluate the performance of our hearing-range-based algorithm that localizes the jammer using the least-squares approach (LSQ), and we compare it with the virtual force iterative localization algorithm (VFIL) from our prior work [4] under various network conditions, including different network node densities, jammer's NLB radii, and the locations of the jammer in the network. To make a fair comparison, we adopted a version of the VFIL algorithm that also does not rely on the information of the jammer's NLB just as LSQ. Furthermore, we tested both LSQ and VFIL algorithms on the same set of network topologies.

To study the impact of the node distribution on both algorithms, we choose two representative network deployments: simple deployment and smart deployment. The nodes in the simple deployment follow a uniform distribution, corresponding to a random deployment, e.g., sensors are randomly disseminated to the battlefield or the volcano vent. Nodes may cluster together at some spots while may not cover other areas, as shown in Figure 5(a). The smart deployment involves carefully placing nodes so that they cover the entire deployment region well and the minimum distance between any pair of nodes is bounded by a threshold, as shown in Figure 5(b). This type of deployment can be achieved using location adjustment strategies [11] after deployment.

In total, we generated 1000 network topologies in a 300-by-300 meter region for each deployment. The normal communication range of each node was set to 30 meters. Unless specified, we placed the jammer at the center of the network, $(0, 0)$, and set the jammer's NLB to 60 meters.

To evaluate the accuracy of localizing the jammer, we define the localization error as the Euclidean distance between the estimated jammer's location and the true location. To capture the statistical characteristics, we studied the average errors under multiple experimental rounds and we presented both the means and the Cumulative Distribution Functions(CDF) of the localization error.

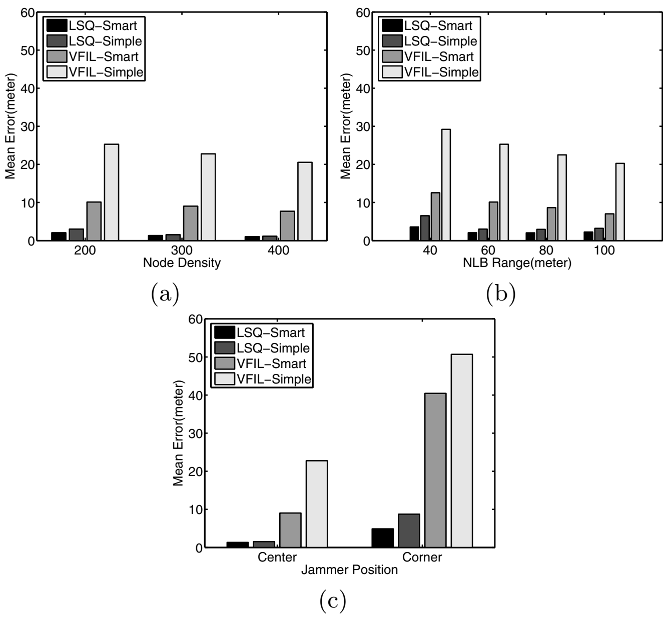


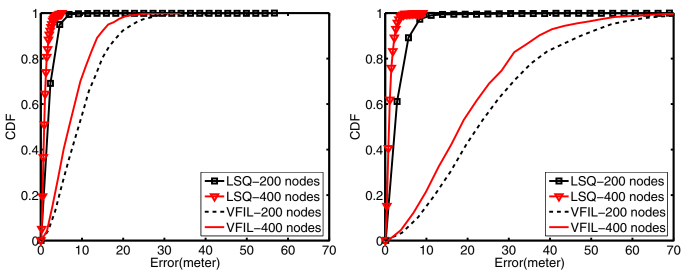
Fig. 6. The impact of various factors on the performance of LSQ and VFIL algorithms: (a) node density; (b)jammer's NLB range; (c) jammer's position in the network

4.2 Performance Evaluation

Impact of the Node Density. We first investigated the impact of the node density on the performance of both the LSQ and VFIL methods. To adjust the network node density, we varied the total number of nodes deployed in the 300-by-300 meter region in the simulation. In particular, we chose to run the experiments on the networks of 200, 300 and 400 nodes, respectively.

We depicted the mean errors for both LSQ and VFIL in Figure 6 (a). Firstly, we observed that LSQ outperformed VFIL consistently in all node densities and node deployment setups. The LSQ's mean errors fall between 1 meter and 3 meters, much smaller than the errors of VFIL, which ranges from 9 to 25 meters. The performance difference can be explained as the following: The VFIL algorithm iteratively searches for the estimated jammer's location until it finds one such that under the assumption of a jammer resided there the derived nodes' categories match with their true categories, e.g., unaffected, jammed, or boundary. Thus, such an estimation is only good-enough but not optimal. In comparison, the LSQ algorithm calculates the location that minimizes all hearing range estimation errors at one step.

Secondly, with the increasing network node densities, the performance of both algorithms improves. Since both algorithms rely on the number of affected nodes to improve the estimation accuracy, the higher the densities, the smaller the mean estimation errors.



(a) Smart (b) Simple

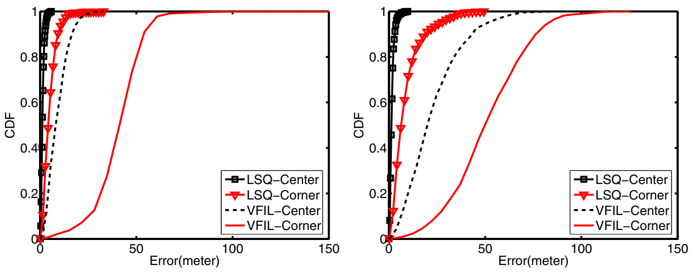
Fig. 7. Cumulative Distribution Function (CDF) of the localization errors under different node densities

Finally, both algorithms performed better in a smart deployment than a simple deployment. In a simple deployment, nodes were not evenly distributed. Thus, when a jammer was placed within an area sparsely covered, without enough affected nodes to provide constraints, the accuracy of the jammer's location estimation suffered. In contrast, the nodes in a smart deployment covered the entire network region evenly, and they supplied reasonable amount of information for the algorithms to localize the jammer. Therefore, both algorithms achieved better localization accuracy in a smart deployment.

We also provided a view of Cumulative Distribution Function (CDF) curves for both algorithms in Figure 7. To make the plot readable, we showed the results of 200 and 400 node cases, omitting the almost overlapped 300-node result. Again, we observed that the LSQ outperformed VFIL constantly. Particularly, under the smart deployment, 90% of the time LSQ can estimate the jammer's location with an error less than 4.2 meters, while VFIL can only achieve 18.8 meters 90% of the time, resulting in an improvement of 80%. While under a simple deployment, LSQ improved the localization accuracy by 95%, as its estimation errors were less than 5.9 meters 90% of the time versus 47.5 meters for VFIL.

Impact of the Jammer's NLB Range. To study the effects of various jammer's NLB ranges to the localization performance, we examined networks with 200 nodes and set the jammer's NLB radius to 40m, 60m, 80m and 100m, respectively. The results were plotted in Figure 6(b) showing that the LSQ method still largely outperformed the VFIL method by over 60%. Additionally, we noticed that the localization errors of VFIL decreased linearly when the jammer's NLB range increased. However, the errors of LSQ only lessened when the NLB range increased from 40m to 60m and became steady afterwards. This is because the number of affected nodes in the 40m NLB range scenario was not enough for LSQ to localize the jammer accurately, i.e., the number of equations that can be created for the LSQ algorithm was not enough. When the NLB range became large enough (e.g., larger than 60m), the LSQ algorithm had enough equations to produce estimation with similar average errors.

Impact of the Jammer's Position. We investigated the impact of the jammer's position by placing it at the center $(0, 0)$ and at the corner $(130, -130)$,



(a) Smart (b) Simple

Fig. 8. Comparison of Cumulative Distribution Function (CDF) of the localization errors when the jammer is placed in the center or at the corner

respectively. In both cases, we set the jammer' NLB range to 60m and used 300-node networks.

Figure 6(c) shows that the performance of both LSQ and VFIL degraded when the jammer is at the corner of the network. Because the affected nodes were located on one side of the jammer, causing the estimated location biased towards one side. However, in both simple and smart deployments, LSQ still maintained a localization error less than 10m, which is 1/3 of a node's transmission range. VFIL produced errors of more than 40m when the jammer was at the corner, making the results of jammer localization unreliable. Thus, LSQ is less sensitive to the location of the jammer. The observations of the CDF results in Figure 8 provide a consistent view with the mean errors.

5 Conclusion

We focused this work on addressing the problem of localizing a jammer in wireless networks. We proposed a hearing-range-based localization algorithm that utilizes the changes of network topology caused by jamming to estimate the jammer's location. We have analyzed the impact of a jammer and have shown that the levels of the nodes' hearing range changes are determined quantitatively by the distance between a node to the jammer. Therefore, we can localize the jammer by estimating the new hearing ranges and solving a least-squares problem. Our approach does not depend on measuring signal strength inside the jammed area, nor does it require to deliver information out of the jammed area. Thus, it works well in the jamming scenarios where network communication is disturbed. Additionally, compared with prior work which involves searching for the location of the jammer iteratively, our hearing-range-based algorithm finishes the location estimation in one step, significantly reducing the computation cost while achieving better performance. We compared our approach with the virtual force iterative localization algorithm (VFIL) in simulation. In particular, we studied the impact of node distributions, network node densities, jammer's transmission ranges, and jammer's positions on the performance of both algorithms. Our extensive simulation results have confirmed that the hearing-range-based algorithm is effective in localizing jammers with high accuracy and outperforms VFIL algorithm in all experiment configurations.

References

1. Bahl, P., Padmanabhan, V.N.: RADAR: An in-building RF-based user location and tracking system. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), pp. 775–784 (March 2000)
2. Cagalj, M., Capkun, S., Hubaux, J.: Wormhole-Based Anti-Jamming Techniques in Sensor Networks. IEEE Transactions on Mobile Computing, 100–114 (January 2007)
3. Chen, Y., Francisco, J., Trappe, W., Martin, R.P.: A practical approach to landmark deployment for indoor localization. In: Proceedings of the Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON (2006)
4. Liu, H., Xu, W., Chen, Y., Liu, Z.: Localizing jammers in wireless networks. In: Proceedings of IEEE PerCom International Workshop on Pervasive Wireless Networking, IEEE PWN (2009)
5. Ma, K., Zhang, Y., Trappe, W.: Mobile network management and robust spatial retreats via network dynamics. In: Proceedings of the The 1st International Workshop on Resource Provisioning and Management in Sensor Networks, RPMSN 2005 (2005)
6. Noubir, G., Lin, G.: Low-power DoS attacks in data wireless lans and countermeasures. SIGMOBILE Mob. Comput. Commun. Rev. 7(3), 29–30 (2003)
7. Pelechrinis, K., Koutsopoulos, I., Broustis, I., Krishnamurthy, S.V.: Lightweight jammer localization in wireless networks: System design and implementation. In: Proceedings of the IEEE GLOBECOM (December 2009)
8. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys 2004: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 95–107 (2004)
9. Priyantha, N., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom), pp. 32–43 (August 2000)
10. Proakis, J.G.: Digital Communications, 4th edn. McGraw-Hill, New York (2000)
11. Wang, G., Cao, G., Porta, T.L.: Movement-assisted sensor deployment. IEEE Transactions on Mobile Computing 5(6), 640–652 (2006)
12. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems 10(1), 91–102 (1992)
13. Wood, A., Stankovic, J., Son, S.: JAM: A jammed-area mapping service for sensor networks. In: 24th IEEE Real-Time Systems Symposium, pp. 286–297 (2003)
14. Xu, W., Trappe, W., Zhang, Y.: Channel surfing: defending wireless sensor networks from interference. In: IPSN 2007: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 499–508 (2007)
15. Xu, W., Trappe, W., Zhang, Y., Wood, T.: The feasibility of launching and detecting jamming attacks in wireless networks. In: MobiHoc 2005: Proceedings of the 6th ACM International Symposium on Mobile Ad Noc Networking and Computing, pp. 46–57 (2005)
16. Xu, W.: On adjusting power to defend wireless networks from jamming. In: Proceedings of the Fourth Annual International Conference on Mobile and Ubiquitous Systems, MobiQuitous (2007)

Self-stabilizing Synchronization in Mobile Sensor Networks with Covering

Joffroy Beauquier[1,⋆] and Janna Burman[2,⋆⋆]

[1] University Paris Sud, LRI, UMR 8623, Orsay, F-91405
jb@lri.fr

[2] Dept. of Industrial Engineering & Management, Technion, Haifa 32000, Israel
bjanna@technion.ac.il

Abstract. Synchronization is widely considered as an important service in distributed systems which may simplify protocol design. *Phase clock* is a general synchronization tool that provides a form of a logical time. This paper presents a *self-stabilizing* (a tolerating state-corrupting transient faults) phase clock algorithm suited to the model of *population protocols with covering*. This model has been proposed recently for sensor networks with a very large, possibly *unknown* number of *anonymous* mobile agents having *small memory*. Agents interact in pairs in an *asynchronous* way subject to the constraints expressed in terms of the *cover times* of agents. The cover time expresses the "frequency" of an agent to communicate with all the others and abstracts agent's communication characteristics (e.g. moving speed/patterns, transmitting/receiving capabilities). We show that a phase clock is impossible in the model with only constant-state agents. Hence, we assume an existence of resource-unlimited agent - the base station.

The clock size and duration of each phase of the proposed phase clock tool are adjustable by the user. We provide application examples of this tool and demonstrate how it can simplify the design of protocols. In particular, it yields a solution to Group Mutual Exclusion problem.

Keywords: population protocols, self-stabilization, cover time, synchronization, phase clock.

1 Introduction

Recently, attempts have been made for developing a model for mobile sensor networks. In 2004, Angluin et al. [1] proposed the model of population protocols (PP) of very limited mobile agents. In this model, anonymous finite-state agents move in an asynchronous way and can communicate and exchange information in pairs when they come into range of each other. One of the goals of

⋆ The work of this author was partially supported by grants from Grand Large project, INRIA Saclay.

⋆⋆ The work of this author was partially supported by grants from the Israel Council for Higher Education and from the Israel Science Foundation.

studying the population protocol model was to determine what is computable with minimal assumptions about a mobile ad-hoc network. For that reason, the agents are asynchronous, anonymous, have a small memory and no assumptions are made on the size of the population or on the way they move, except for a fairness assumption ensuring that any continuously reachable global configuration is eventually reached. It was shown in [3] that the set of applications that can be solved in the original population protocol model of [1] is rather limited. Hence, various extensions were suggested. Some of them are: assumption of a distinguishable resource-unlimited agent - the base station [10], unique identifiers [18], probabilistic scheduler and unique leader agent [2]. In [17], an oracle for eventual leader detection is assumed, and even with the help of the oracle, it is shown that constructing uniform self-stabilizing leader election in (communication) rings is impossible when local fairness is used (somewhat weaker fairness than in [1]). In [6], Beauquier et al. propose to add to the population protocol model a notion of "speed" which provides a stronger fairness. With this notion, each agent meets the other agents with a quantitative constraint expressed in terms of its *cover time* (hence, the term *covering*). The cover time of an agent is the minimum number of events in the whole system for an agent to have met with each other agent with certainty. The "faster" an agent is, the lesser is its cover time. Cover times make it possible to construct fast converging deterministic protocols and evaluate their complexities (it is impossible with a simple fairness assumption as in [1]).

The model of population protocols with covering suits well sensor networks with agents that move in a more predictable manner (so that they are likely to imply deterministic cover times). E.g., sensor networks in a production line; meteorological sensor networks of intercommunicating agents hosted by radiosondes / balloons / satellites cruising at different altitudes or kinds of an orbit (see, e.g, [25,26]); EMMA project [24]- a pollution monitoring network of sensors attached to different kinds of public transport vehicles (moving according to different itineraries).

The population protocol model with covering has been studied in [8]. It presents an automatic transformer that converts an algorithm (requiring an initialization) into its self-stabilizing version (that operates correctly from any initial state). We take a step further and consider the problem of phase synchronization in systems of mobile agents. We propose (see Sec. 3) a *phase clock algorithm* (see, e.g., [20,14]) that provides an infinite repetition of phases, $0, 1, \ldots, \mathbf{K}-1$, where each phase may be associated with some number of steps of the distributed agents. A phase clock in an asynchronous model requires that any agent x does not execute phase $i+1$ before all the agents that can directly communicate with x have executed phase i $(\mod \mathbf{K})$. To implement the phase clock, each agent is allocated with a constant size ($\mathbf{K} > 2$) counter, the *clock*, that represents the current phase number at the agent. Refer to Sec. 3 for the specification of the phase clock.

In Sec. 4, we give several examples to demonstrate how a phase clock can be used to solve synchronization problems, in the population protocol model. First, we present a simple multi-phase design of a Majority Consensus problem. Then, we easily solve a version of the Group Mutual Exclusion problem. Finally, we propose a synchronous model framework to design specific protocols. According to this framework, a protocol is designed in a simplified synchronous version of the PP model with covering. When designed, the protocol can be executed in the original asynchronous model in combination with a *synchronizer* algorithm [4] (that we construct and) that provides an appropriate synchronization. We show that this framework is useful in designing resource consuming tasks.

The algorithms presented in this paper are *self-stabilizing*. Such algorithms have the important property of operating correctly regardless of their initial state (except for some bounded period). In practice, self-stabilizing algorithms adjust themselves automatically to any changes or corruptions of the network components (excluding the algorithm's code). These changes are assumed to cease for some sufficiently long period. Self-stabilization is considered here for two reasons. First, mobile agents are generally fragile, subject to failures and hard to initialize. Second, systems of mobile agents are by essence dynamic, some agents leave the system while new ones are introduced. Self-stabilization is a well adapted framework for dealing with such situations. For instance, when two groups of agents merge, the clock values in one group can be very different from the values in the other group. A self-stabilizing solution will bring them to a correct global configuration, in which there may be only a "small" acceptable gap between the clocks. Self-stabilizing phase clocks for conventional asynchronous communication models are given, e.g., in [14,5,19,11]. To the best of our knowledge, this is the first time a phase clock is presented in the population protocol model.

We show (Theorem 1, Sec. 3) that it is impossible to implement a phase clock in the PP model with covering when every agent has only a constant number of states (independent of the population size). In particular, it implies the same impossibility result in the original PP model. Thus, to be able to design a phase clock, we assume (as in [10,6,8]) a resource-unlimited agent, the base station (BS). Note that this is a rather natural assumption, because an agent such as BS is present in many sensor network applications.

Note also that constructing a phase clock, in which the agents synchronize their clocks only when they meet BS, is relatively easy. However, it is not efficient, because the synchronization of one phase takes $\Omega(\mathbf{cv_{max}})$ meetings (events), where $\mathbf{cv_{max}}$ is the maximum cover time. The solution we propose is more efficient, because it uses the fastest agents to propagate the clock values. In this solution, the synchronization of a phase takes $\Omega(\mathbf{cv_{min}})$ meetings, where $\mathbf{cv_{min}}$ is the minimum cover time, which can be considerably smaller than $\mathbf{cv_{max}}$.

2 The Model

Transition System

Let A be the set of *all* the agents in system $\mathcal{S}$, where $|A| = \mathbf{n}$ is unknown to agents. The *base station* (BS) is a distinguishable, resource-unlimited and (usually) non-mobile[1] agent. All the other agents are finite-state, anonymous (no identifiers and uniform codes) and are referred in the paper as *mobile*.

[1] If BS is mobile, it does not change the analysis in this paper.

Population protocols can be modeled as transition systems. We adopt the common definitions of the following: *state* of an agent (a vector of the values of its variables), *configuration* (a vector of states of all the agents), *transition* (atomic step of *two* communicating agents and their associated state changes), *execution* (a possibly infinite sequence of configurations related by transitions), *terminal configuration* (in which no applicable transition can change the configuration), *infinite / finite execution* (that does not contain / contains a terminal configuration). When a terminal configuration is reached, we say that *termination* has occurred. For formal definitions, refer, e.g., to [27].

An *event* (x, y) is a pairwise communication (meeting) of two agents x and y. Without loss of generality, we assume that no two events happen simultaneously. Each execution corresponds to a unique sequence of events. The *length of a (finite) execution* is the minimum number of events until the termination. The *event complexity* of an algorithm is the maximum length of an execution until termination. For some l ($\in \mathbb{N}_0$) and agent x, let $[l]^x$, *l local events at x*, be l consecutive (from x's point of view) events in which agent x participates. This stands in contrast to l *global events* (or just events) which are l consecutive events in an execution. Note that if $[l]^x$ events occurred, then at least l global events occurred.

Intuitively, it is convenient to view executions as if a *scheduler* (an adversary) "chooses" which two agents participate in the next event. Formally, a scheduler $\mathcal{D}$ is a predicate on the sequences of events. A *schedule* of $\mathcal{D}$ is a sequence of events that satisfies predicate $\mathcal{D}$.

A protocol is called *self-stabilizing* if starting from an arbitrary configuration, it reaches a *legal configuration* (defined by the problem specification) eventually and remains in legal configurations thereafter. When this happens, we say that *stabilization* has occurred. The maximum number of events until stabilization is called the *stabilization time* or *stabilization event complexity* of the protocol.

Cover Time Property (Covering)
Given $\mathbf{n}$ agents, a vector $\overline{\mathbf{cv}} = (\mathbf{cv}_1, \mathbf{cv}_2, \ldots, \mathbf{cv}_\mathbf{n})$ of positive integers (the *cover times*) and a scheduler $\mathcal{D}$, we say that $\mathcal{D}$ (as well as each of its schedules) satisfies the *cover time property*, if in any $\mathbf{cv}_i$ ($i \in \{1 \ldots \mathbf{n}\}$) consecutive events of each schedule of $\mathcal{D}$, agent i meets every other agent at least once.[2] Any *execution* of a system under such a scheduler is one that *satisfies the cover time property.*
For two agents x and y, if $\mathbf{cv}_x < \mathbf{cv}_y$, then x is said to be *faster* than y, and y *slower* than x. The minimum cover time value is denoted by $\mathbf{cv}_{\mathbf{min}}$ and the maximum one by $\mathbf{cv}_{\mathbf{max}}$. A *fastest/slowest* agent z has $\mathbf{cv}_z = \mathbf{cv}_{\mathbf{min}}/\mathbf{cv}_z = \mathbf{cv}_{\mathbf{max}}$.
Remark 1. Note that there are vectors of integers such that there is no possible schedule satisfying the cover time property implied by the vector (e.g, $\overline{\mathbf{cv}} = (4, 6, 10, 15)$). From now on, we assume $\overline{\mathbf{cv}}$s implying at least one possible schedule.

Agents are Not Assumed to Know Cover Times. Instead, we do assume that when two agents meet, they are able to detect which of them is faster (unless none of them is).[3] One way to quantify that, is to assume that each agent

[2] We emphasize that this definition does *not* imply that an agent knows its cover time.
[3] Even, a weaker assumption is possible. See Remark 2, Sec. 3.

x is given with a *category* number $\mathbf{cat}_x$ (a positive integer). For instance, different kinds of public transport vehicles (moving according to different itineraries) in the EMMA project [24] can correspond to different categories. In ZebraNet [23] (a habitat monitoring application where sensors are attached to zebras), a measured temperature (or pulse) far from the normal can imply an ill animal, that is slower. Hence, this animal is assigned a category number that is bigger than the one of healthy animals. The number of different categories, $\mathbf{m}$, is generally much smaller than the size of the population $\mathbf{n}$ ($\mathbf{m} \ll \mathbf{n}$) and agents do not know the value of $\mathbf{m}$. Note that categories are not identifiers, because there can be an arbitrary number of agents in the same category and because agents in the same category are indistinguishable.

For BS, we need the following stronger assumption to be able to use the algorithms of [8]. We assume that BS, but not a mobile agent, is able to estimate an *upper bound* of $\mathbf{cv}$ of an agent *it meets.* Recall that BS is resource-unlimited, what may help it in this task. E.g., BS may maintain a table providing an upper bound on the cover time of each category.

3 Self-stabilizing Phase Clock

Our objective is to design a self-stabilizing bounded (using a constant amount of memory at every mobile agent) phase clock in the model of population protocols with covering. To state the problem, we use a conventional definition of a phase clock (see, e.g., [14,5]) that we adapt and somewhat generalize (see the *Frequency of progress* condition below) to suit better the model we use. In Fig. 1, we present a self-stabilizing phase clock algorithm which is later proven to satisfy the following specification, after a certain stabilization time has elapsed.

Definition 1. *[Bounded Phase Clock Problem Specification]*
A phase clock provides each agent with a clock/phase number *subject to the following conditions. Let the clock number of an agent x be stored in a variable* $\texttt{clock}_x$.
Progress: *In any execution, every variable* `clock` *is updated infinitely often and each time, according to the assignment statement* $\texttt{clock} := (\texttt{clock} + 1) \mod \mathbf{K}$ *only.*
Frequency of progress: *In any execution, after every update of* $\texttt{clock}_x$*, the next update cannot happen before* $\beta(\texttt{clock}_x)$ *global events, where β is a function that maps each phase number to a lower bound on the duration of the phase, which is counted by the number of global events. It is the responsibility of the designer, who uses the phase clock, to define the required number* $\mathbf{K}$ *of phases and the value of β for every phase.*
Asynchronous unison: *In any configuration reached by an execution, the clock numbers of any two agents that can meet differ by no more than 1 (* $\mod \mathbf{K}$*). That is, for any two agents x and y that can meet, the following predicate* in_phase(x, y) *is true.*

$$in\_phase(x,y) \equiv \texttt{clock}_x = \texttt{clock}_y \bigvee \texttt{clock}_x = (\texttt{clock}_y + 1) \mod \mathbf{K} \bigvee \texttt{clock}_y = (\texttt{clock}_x + 1) \mod \mathbf{K}$$

Note that since in our model every pair of agents can meet sometime, if the *Asynchronous unison* condition holds, the clock numbers of *any* two agents differ by no more than 1 ($\mod \mathbf{K}$).

We start by proving that if every agent has only a constant number of states, independent of $\mathbf{n}$ (there is no resource-unlimited BS), the conditions of the specification of a phase clock cannot be realized by any system in the PP model with covering. Hence, the usage of BS.

To prove the following Theorem 1, we adopt the method which is used in [9] (the complete version of [8]) to prove a similar impossibility result (see Theorem 4.1, in [9]). We adjust the proof in [9] to suit the case of a phase clock. For completeness, the adjusted and complete proof is provided in the appendix.

Definition 2. *Assume the model of population protocols with covering.*

- *Let a* generic solution *be an algorithm that outputs a transition system for every possible population of (every) size* $\mathbf{n}$ *and for (every) vector of cover times* $\overline{\mathbf{cv}}$.[4]
- *Let a* generic solution for a phase clock *be a generic solution providing only transition systems, whose executions satisfy the conditions of the phase clock specification in Def. 1.*
- *A* local transition system of an agent x *is a projection of the (global) transition system (defined in Sec. 2) on* x. *That is, it is the set of all the states and the transitions of* x. *Since the codes of the mobile agents are uniform, their local transition systems are identical. A* (global) transition system is bounded, *if and only if the two local transition systems of a mobile agent and of BS are bounded in size, independently of* $\mathbf{n}$.
- *A* generic solution is bounded, *if and only if every (global) transition system provided by this solution is bounded.*

Theorem 1. *Assume the model of population protocols with covering. Consider Def. 1 and let* $\mathbf{K} \geq 6$. *Then, even if there is no fault of any kind and even if the clocks of all the agents (in Def. 1) are initialized to the same value at once, no bounded generic solution for a phase clock exists.*

Self-Stabilizing Bounded Phase Clock - Fig. 1

Every agent x has a clock variable $\texttt{clock}_x$ that indicates the ongoing phase number. Agents synchronize their clocks with BS with the help of the fastest agents (whenever it is efficient), and not in a trivial way - only on the meetings with BS. This improves the event complexity of clock synchronization at each phase (as explained in Sec. 1). As a result, it also improves the stabilization time of the phase clock.

[4] Recall that we assume only $\overline{\mathbf{cv}}$s implying at least one possible schedule (Remark 1).

BS locally counts the required number of events for each phase p (a function of $[\beta(p)]^{BS}$) and then, increases the clock by one (mod $\mathbf{K}$). The fastest agents are responsible to propagate the clock value from BS to other agents. At most $2 \cdot \mathbf{cv_{min}}$ events are required for this propagation - $\mathbf{cv_{min}}$ events for the propagation from BS to the fastest agents and then, another $\mathbf{cv_{min}}$ events for the propagation from the fastest agents to others.[5] To implement this propagation, the phase clock has to determine the fastest agents in a self-stabilizing way. Thus, there is a $\texttt{fastest}_x$ bit, for every agent x.

For the purpose of event counting (and in particular, to evaluate the β values), BS should be able to estimate the upper bounds on the values of $\mathbf{cv_{min}}$ and $\mathbf{cv_{max}}$. The self-stabilizing algorithms that estimate $\mathbf{cv_{min}}$ and $\mathbf{cv_{max}}$ are presented in [8]. They are executed at BS and stabilize in $O([\mathbf{cv_{min}}]^{BS})$ events. We assume that those algorithms provide the upper bounds of $\mathbf{cv_{min}}$ and $\mathbf{cv_{max}}$ in variables $\mathbf{cv}^*_{\texttt{min}}$ and $\mathbf{cv}^*_{\texttt{max}}$ respectively.

In addition, note that, if $\mathbf{K} > 2$, our phase clock satisfies the *Frequency of progress* condition with an upper bound of $[\beta(\texttt{clock}_x)+2\cdot\min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}$ events, for any agent x. That is, the next update of $\texttt{clock}_x$ (that cannot happen before $\beta(\texttt{clock}_x)$ global events, according to *Frequency of progress* condition) is done after at most $[\beta(\texttt{clock}_x) + 2 \cdot \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}$ events. It is proven in Lem. 6, below.

The analysis below (Theorem 2) shows that the stabilization time of the presented phase clock is $O(\ [\max_p(\beta(p)) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}\ )$ *local* events (at BS), where $p \in \{0, \dots \mathbf{K} - 1\}$.
Let us express this complexity in terms of the global events. By the cover time property (see Sec. 2), in any $\mathbf{cv}_{BS}$ global events, BS participates in at least one event with every other agent out of $\mathbf{n} - 1$. Hence, in any $\mathbf{cv}_{BS}$ global events, BS counts locally at least $\mathbf{n} - 1$ events. Thus and since $\mathbf{cv}_{BS} \leq \mathbf{cv_{max}}$, the stabilization time of the phase clock is equivalent to $O(\ \frac{\mathbf{cv_{max}}}{\mathbf{n}-1} \cdot (\max_p(\beta(p)) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}}))\ )$ *global* events.

Note that in the following analysis, we assume that the system is started in an arbitrary configuration, but then, no faults or population changes occur until stabilization. This is a common assumption done during analysis of self-stabilizing algorithms.

Definition 3 (Complete and Incomplete phase).
A phase p ($\in \{0, \dots \mathbf{K}-1\}$) is a segment of an execution of the phase clock (Fig. 1) during which $\texttt{clock}_{\texttt{BS}} = p$. *A* complete phase *$p$ is a phase p that starts after an event where line 3 is executed and ends during an upcoming later event where line 3 is executed. An* incomplete phase *p is a phase p which is not complete (incomplete phases arise from a bad (faulty) initialization).*

Lemma 1. *Every complete phase p lasts $[\beta(p) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}$ events, which correspond to at least $\beta(p) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})$ global events (which are*

[5] By the code of the phase clock, if $2 \cdot \mathbf{cv_{min}} > \mathbf{cv_{max}}$, agents synchronize their clocks faster (in at most $\mathbf{cv_{max}}$ events), on the meetings with BS. Thus, clock synchronization at each phase takes $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events.

```
Memory in a mobile agent $x \neq$ BS:
  $\texttt{clock}_x \in \{0,\ldots,\mathbf{K}-1\}$              /* the clock indicator of $x$ */
  $\texttt{fastest}_x \in \{0,1\}$                 /* indicates if $x$ is a fastest agent */
  $\mathbf{cat}_x$ : positive integer                      /* category number of $x$ */

Memory in BS:
  event_ctr : non-negative integer          /* counter of the local events at BS */
  $\texttt{clock}_{\texttt{BS}} \in \{0,\ldots,\mathbf{K}-1\}$
  $\mathbf{cat}^*_{\texttt{min}}$ : positive integer           /* minimum category estimated by BS */
  $\texttt{t\_}\mathbf{cat}^*_{\texttt{min}}$ : positive integer     /* temporal minimum category estimated by BS */
  $\mathbf{cv}^*_{\texttt{min}}$, $\mathbf{cv}^*_{\texttt{max}}$ : positive integer  /* estimated $\mathbf{cv_{min}}$ and $\mathbf{cv_{max}}$ provided externally */

  /* phase - duration mapping $\beta$ */
  $\beta : \{0,\ldots,\mathbf{K}-1\} \rightarrow PD$,
              where $PD$ is the set of non-negative functions of $\mathbf{cv}^*_{\texttt{min}}$ and $\mathbf{cv}^*_{\texttt{max}}$

  $legal(\ clk_f, clk_s\ ) \equiv (\ clk_f = clk_s \bigvee \texttt{clock}_f = (\texttt{clock}_s + 1) \mod \mathbf{K}\ )$

Whenever agent $x$ communicates with BS:
1  $\texttt{event\_ctr} := \min(\ \texttt{event\_ctr},\ \beta(\texttt{clock}_{\texttt{BS}}) + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})\ ) - 1$
2  if $\texttt{event\_ctr} = 0$ then                          /* switch to the next phase */
3     $\texttt{clock}_{\texttt{BS}} := (\texttt{clock}_{\texttt{BS}} + 1) \mod \mathbf{K}$;  $\texttt{event\_ctr} := \beta(\texttt{clock}_{\texttt{BS}}) + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})$
4     $\mathbf{cat}^*_{\texttt{min}} := \texttt{t\_}\mathbf{cat}^*_{\texttt{min}}$;  $\texttt{t\_}\mathbf{cat}^*_{\texttt{min}} := \mathbf{cat}_x$
5  if $(\mathbf{cat}_x = \mathbf{cat}^*_{\texttt{min}})$ then $\texttt{fastest}_x := 1$        /* $x$ is one of the fastest agents */
6  else $\texttt{fastest}_x := 0$
7  $\texttt{t\_}\mathbf{cat}^*_{\texttt{min}} := \min(\mathbf{cat}_x, \texttt{t\_}\mathbf{cat}^*_{\texttt{min}})$
8  $\texttt{clock}_x := \texttt{clock}_{\texttt{BS}}$

Whenever agent $x$ communicates with an agent $y \neq$ BS:
9   if $\mathbf{cat}_y > \mathbf{cat}_x$ then $\texttt{fastest}_y := 0$                /* $y$ is a non-fastest agent */
10  if $(\ \texttt{fastest}_x \wedge \neg\texttt{fastest}_y\ )$ then
11     if $(\ legal(\ \texttt{clock}_x, \texttt{clock}_y\ ) \vee \neg legal(\ \texttt{clock}_y, \texttt{clock}_x\ )\ )$ then
12        $\texttt{clock}_y := \texttt{clock}_x$
```

Fig. 1. Self-stabilizing Bounded Phase Clock

at least $\min(2\cdot\mathbf{cv_{min}}, \mathbf{cv_{max}})$ *events). Every incomplete phase lasts less than* $[\max_i\{\beta(i)\} + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}$ *events.*

Proof: The lemma follows from lines 1-3 and the definition of β (Fig. 1). ∎

Lemma 2. *After at most* $[2\cdot(\ \max_p(\beta(p)) + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})\ )]^{BS} + \mathbf{cv_{min}}$ *events, all* `fastest` *bits are correct. That is, for every fastest mobile agent* f, $\texttt{fastest}_f = 1$ *and for every non-fastest mobile agent* s, $\texttt{fastest}_s = 0$.

Proof: By line 1, in at most $[\max_p(\beta(p)) + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})]^{BS}$ events, the condition at line 2 is true and lines 3, 4 are executed. After an execution of line 4, $\texttt{t\_}\mathbf{cat}^*_{\texttt{min}}$ is greater than or equal to the minimum category value (the category value of a fastest agent). Then, during the next $[\mathbf{cv}^*_{\mathtt{min}}]^{BS}$ events, due to the cover time property, a fastest agent meets BS and $\mathtt{t\_}\mathbf{cat}^*_{\mathtt{min}}$ is assigned the minimum category value, at line 7. Then, later, the condition at line 2 becomes true again and $\mathbf{cat}^*_{\mathtt{min}}$ is assigned a correct minimum category too. Thus, $\mathbf{cat}^*_{\mathtt{min}}$ is correct in no more than $[2 \cdot (\ \max_p(\beta(p)) + \min(2 \cdot \mathbf{cv}^*_{\mathtt{min}}, \mathbf{cv}^*_{\mathtt{max}})\ )]^{BS}$ events. Then, the same process is repeated in each phase and $\mathbf{cat}^*_{\mathtt{min}}$ is assigned the same correct value each time line 4 is executed. From the moment when $\mathbf{cat}^*_{\mathtt{min}}$ is correct, in at most additional $\mathbf{cv_{min}}$ events, every fastest agent f meets BS and its $\mathtt{fastest}_f$ bit is set correctly to 1, at line 5. Since $\mathbf{cat}^*_{\mathtt{min}}$ value stays unchanged, $\mathtt{fastest}_f$ stays unchanged too. In addition, from the moment when $\mathbf{cat}^*_{\mathtt{min}}$ is correct, in at most $\mathbf{cv_{min}}$ events, a fastest agent f meets every non-fastest mobile agent s and its $\mathtt{fastest}_s$ bit is set correctly to 0, at line 9. Since $\mathbf{cat}^*_{\mathtt{min}}$ value stays unchanged, $\mathtt{fastest}_s$ stays unchanged too (line 5 is not executed for s). ∎

Remark 2. Correctness of Lem. 2 relays on the assumption that when two agents meet, they correctly detect which of them is faster (using the categories; see Sec. 2). In [7], we address the case where there could be mistakes in these detections. But, as long as the $\mathtt{fastest}$ bits still stabilize, the phase clock stabilizes too. This fact, allows to use the phase clock even with weaker assumptions than those stated in Sec. 2. Though, in case of such mistakes, the phase clock may stabilize after a larger number of events.

Lemma 3 (Asynchronous unison). *After at most* $[2 \cdot (\ \max_p(\beta(p)) + \min(2 \cdot \mathbf{cv}^*_{\mathtt{min}}, \mathbf{cv}^*_{\mathtt{max}})\ )]^{BS} + \mathbf{cv_{min}} + 2 \cdot \min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ *events, for any two agents x and y, in_phase(x, y) is true.*

Proof: First, note that $\mathtt{clock_{BS}}$ can be updated only at line 3. If $\mathtt{clock_{BS}}$ changes in less than $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ global events, then by Lem. 1, $\mathtt{clock_{BS}}$ stays unchanged during at least the next $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events. Now, assume that $\mathtt{clock_{BS}}$ and the $\mathtt{fastest}$ bits stay unchanged during at least the next $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events in a phase p and that the $\mathtt{fastest}$ bits are correct (as in Lem. 2). We now show that in additional $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events, the lemma holds. Whenever an agent x meets BS, it assigns $\mathtt{clock}_x := \mathtt{clock_{BS}}$. By the condition at line 10, a fastest agent cannot execute line 12. Thus, in at most $\mathbf{cv_{min}}$ events, for every fastest agent f, $\mathtt{clock}_f = \mathtt{clock_{BS}}$. Whenever a fastest mobile agent f meets a non-fastest mobile agent s, by the condition at line 11, if either $\mathtt{clock}_s = \mathtt{clock}_f - 1 \mod \mathbf{K}$, or $\mathtt{clock}_s$ and $\mathtt{clock}_f$ values differ by *more* than 1, then line 12 is executed and the predicate $\mathtt{clock}_s = \mathtt{clock}_f$ becomes true. Thus, if $2 \cdot \mathbf{cv_{min}} > \mathbf{cv_{max}}$, in at most $\mathbf{cv_{max}}$ events, for every agent x, $\mathtt{clock}_x = \mathtt{clock_{BS}}$. Otherwise (if $2 \cdot \mathbf{cv_{min}} < \mathbf{cv_{max}}$), in at most additional $\mathbf{cv_{min}}$ events (after $\mathtt{clock}_f = \mathtt{clock_{BS}}$, for every fastest agent f), a fastest agent f meets every non-fastest mobile agent s. After such meetings occur, for every s, $\mathtt{clock}_s = \mathtt{clock}_f \vee \mathtt{clock}_s = \mathtt{clock}_f + 1 \mod \mathbf{K}$ holds. Thus, after at most $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events (and since $\mathtt{clock_{BS}}$ stays unchanged during these events), for any two agents x and y, *in_phase*(x, y) is true. Until the next complete phase $(p+1) \mod \mathbf{K}$ starts, $\mathtt{clock_{BS}}$ value stays unchanged and hence, every $\mathtt{clock}_f$ stays unchanged too (by lines 10-12 and 8, at this time, the clock number of a fastest agent can be changed only by BS). This and the code lines 10-12 imply that every $\texttt{clock}_s$ stays unchanged too. By a simple induction, after phase $(p+1) \mod \mathbf{K}$ starts, during this phase and any following phase, for any two agents x and y, *in_phase*(x, y) is true. Thus and by Lem. 2, the lemma follows. ∎

Lemma 4 (Liveness). *Every variable* `clock` *is updated infinitely often.*

Proof: Assume that the `fastest` bits are correct (as in Lem. 2). Assume by contradiction that for some agent x, $\texttt{clock}_x$ is not updated starting from some configuration C. Then, by Lem. 1, in at most $[\max_p\{\beta(p)\} + 2 \cdot \mathbf{cv}^*_{\texttt{min}}]^{BS}$ events, $\texttt{clock}_{\texttt{BS}}$ is updated to value j. Thus, x cannot be BS. Consider some fastest agent f with $\texttt{clock}_f = i$. Note that by the condition at line 10, $\texttt{clock}_f$ can be updated only by BS, at line 8. After the update of $\texttt{clock}_{\texttt{BS}}$ and if $i \neq j$, in at most $\mathbf{cv_{min}}$ events, every fastest agent f meets BS and updates its $\texttt{clock}_f$. Otherwise $(i = j)$, $\texttt{clock}_f$ is updated during the next phase $j + 1$. Thus, x cannot be f. After $\texttt{clock}_f$ is updated, in at most $\mathbf{cv_{min}}$ events, f meets any (non-fastest mobile) agent s. By condition at line 11, if $\texttt{clock}_s$ is either smaller than or differs from $\texttt{clock}_f$ by more than 1 ($\mod \mathbf{K}$), then $\texttt{clock}_s$ is updated. Otherwise ($\texttt{clock}_s = \texttt{clock}_f \vee \texttt{clock}_s = \texttt{clock}_f + 1 \mod \mathbf{K}$), $\texttt{clock}_s$ will be updated in the meeting with some fastest agent or BS during the next phase or during the phase that comes after the next one. Thus, x cannot be s either. Hence, a contradiction. ∎

Lemma 5 (Progress). *After at most* $X = [2\cdot(\max_p(\beta(p)) + \min(2\cdot\mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}}))]^{BS} + \mathbf{cv_{min}} + 2\cdot\min(2\cdot\mathbf{cv_{min}}, \mathbf{cv_{max}})$ *events, for every agent x,* $\texttt{clock}_x$ *is updated according to the assignment statement* $\texttt{clock}_x := (\texttt{clock}_x + 1) \mod \mathbf{K}$ *only. After any update of* $\texttt{clock}_x$*, the next update cannot happen before* $\beta(\texttt{clock}_x)$ *global events.*

Proof: Assume that X events occurred. Then, by Lem. 2, all `fastest` bits are correct. By Lem. 3, *in_phase*(x, y) is true, for any two agents x and y. First, consider $\texttt{clock}_{\texttt{BS}}$. By Lem. 1 and the only line of the code where $\texttt{clock}_{\texttt{BS}}$ is updated, the lemma holds for $\texttt{clock}_{\texttt{BS}}$. For any fastest agent f, $\texttt{clock}_f$ can be updated at line 8 only, and only to the value of $\texttt{clock}_{\texttt{BS}}$. Thus, the lemma holds for $\texttt{clock}_f$ as for $\texttt{clock}_{\texttt{BS}}$. For any non-fastest mobile agent s, $\texttt{clock}_s$ can be updated either at line 8, or at line 12. At line 8, it is updated to the value of $\texttt{clock}_{\texttt{BS}}$ and thus, for this update of $\texttt{clock}_s$, the lemma holds. By the condition at line 11, $\texttt{clock}_s$ is updated to $\texttt{clock}_f$ (and in fact, to $\texttt{clock}_{\texttt{BS}}$), only if $\texttt{clock}_f = \texttt{clock}_s + 1 \mod \mathbf{K}$ and thus (by line 12) it is updated according to $\texttt{clock}_s := (\texttt{clock}_s + 1) \mod \mathbf{K}$. In both cases (line 8 or 12), $\texttt{clock}_s$ can be actually updated only to the value of $\texttt{clock}_{\texttt{BS}}$ (recall that $\mathbf{K} > 2$) and in at most $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events since the last update of $\texttt{clock}_{\texttt{BS}}$. By Lem. 1, $\texttt{clock}_{\texttt{BS}}$ is updated again after at least $\beta(\texttt{clock}_{\texttt{BS}}) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}})$ global events. Hence the next update of $\texttt{clock}_s$ cannot happen before $\beta(\texttt{clock}_{\texttt{BS}})$ events that is equal to $\beta(\texttt{clock}_s)$ events. ∎

Theorem 2. *The phase clock in Fig. 1 stabilizes in* $[2 \cdot (\max_p(\beta(p)) + \min(2 \cdot \mathbf{cv}^*_{\texttt{min}}, \mathbf{cv}^*_{\texttt{max}}))]^{BS} + \mathbf{cv_{min}} + 2 \cdot \min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ *events.*

Proof: The theorem directly follows from Def. 1 and lemmas 1-5. ∎

Lemma 6 (Frequency of progress - upper bound). *After stabilization, for every agent x, $\mathtt{clock}_x$ is updated again after no more than $[\beta(\mathtt{clock}) + 2 \cdot \min(2 \cdot \mathbf{cv}^*_{\mathtt{min}}, \mathbf{cv}^*_{\mathtt{max}})]^{BS}$ events.*

Proof: The lemma comes from the fact that only if x is BS, it can execute the assignment $\mathtt{clock}_x := (\mathtt{clock}_x + 1) \mod \mathbf{K}$ (line 3). All the other agents can only adopt the propagated value of $\mathtt{clock}_{\mathtt{BS}}$. In addition, the maximum number of events until the next increase of $\mathtt{clock}_{\mathtt{BS}}$ is $[\beta(\mathtt{clock}_{\mathtt{BS}}) + \min(2 \cdot \mathbf{cv}^*_{\mathtt{min}}, \mathbf{cv}^*_{\mathtt{max}})]^{BS}$ (lines 1-3). Then, after this increase, any x increases its $\mathtt{clock}_x$ value (by the adoption of the propagated value of $\mathtt{clock}_{\mathtt{BS}}$, at lines 8, 12) in no more than $\min(2 \cdot \mathbf{cv}^*_{\mathtt{min}}, \mathbf{cv}^*_{\mathtt{max}})$ events. ∎

The following lemma states an important property that is useful when designing and proving protocols that utilize the phase clock (see Sec.4). First, we define that an event e is the *first event in a phase* p, if and only if in the resulting configuration after e, at least one agent x that has participated in e, is in phase p (that is, $\mathtt{clock}_x = p$) and all the other agents are in phase $p - 1 \mod \mathbf{K}$. By the *Asynchronous unison* condition, a first event occurs at each phase, after stabilization of the phase clock.

Lemma 7. *Consider the first event e in some phase p, after stabilization of the phase clock. During at least $\beta(p)$ previous events to e,* all *the agents are in phase $p - 1 \mod \mathbf{K}$.*

Proof: Let e' be the first event in phase $p - 1$ ($\mod \mathbf{K}$), then during e', there is an agent x participating in e' whose $\mathtt{clock}_x$ is incremented and just before e', the clocks of all the other agents equal to $p - 2 \mod \mathbf{K}$. Hence and by lines 3, 8 and 12, x has to be BS. BS increments its clock to $p - 1$ at line 3. Then, by the cover time property and by lines 8 and 12, in $\min(2 \cdot \mathbf{cv_{min}}, \mathbf{cv_{max}})$ events, the value $p - 1$ of $\mathtt{clock}_{\mathtt{BS}}$ is propagated from BS to all the other agents. Then, all the agents are in phase $p - 1$ and BS stays in this phase for at least additional $\beta(p-1)$ events (by lines 1-4). Thus, all the other agents cannot change their clocks for at least the same $\beta(p - 1)$ events (lines 8, 12). Hence, the lemma holds for e. ∎

4 Examples of Application

Methods of protocol design in the model of population protocols with covering differ from the methods used in other models. Hence, in this section, we provide several examples of how and when the presented phase clock may be useful when designing population protocols.

Designing multi-phase systems
Similarly to [14], the phase clock in this paper may be used to design a multi-phase protocol. An important property that our phase clock provides (see *Frequency of progress* condition) is the ability of an agent to detect when a specific

number of global events has occurred. This may be used to detect the termination/stabilization of distributed tasks performed by the agents. Then, a new task (in a new phase) may be started and executed during the required number of events until the task's termination (and until the next phase starts). To do that, the upper bound on the worst case event complexity W of the task should be known and expressed in terms of $\mathbf{cv_{min}}$ and $\mathbf{cv_{max}}$ (as a non-decreasing function).

For example, we present a simple multi-phase protocol to solve a version of a Majority Consensus problem (MCP) in a self-stabilizing way. In MCP, every agent has an input and output variable. After stabilization, the output variables should contain the majority of all the inputs.[6] The parameters of the phase clock (that we choose for the solution) are as follows: $\mathbf{K} := 3$ and $(\beta(0), \beta(1), \beta(2)) = (\mathbf{cv_{max}}, \mathbf{cv_{min}}, 2 \cdot \mathbf{cv_{min}})$. Hence, the solution alternates 3 phases. The task in phase 0 is to deliver the input values to BS. An agent with $\texttt{clock} = 0$ delivers its input value to BS, only once during the phase. To do that, every mobile agent maintains a `done` bit to indicate if the input value has been already delivered during the current phase. At the end of phase 0, BS calculates the majority of the inputs it has gathered and saves it in its output bit (then, BS erases the inputs to re-initialize for the next phase 0). In phase 1, the `done` bits of the agents are reset, to re-initialize for the next phase 0. Whenever two agents with $\texttt{clock} = 1$ meet, both reset their `done` bits. In phase 2, the majority value is broadcasted from BS to all the other agents by the help of the fastest agents in the following way. Whenever an agent x meets a fastest agent (the fastest agents are determined in the same way as in Fig.[1]) or BS, and both have $\texttt{clock} = 2$, x copies the output value of the other agent into its own output variable.

By Lem. [7], it is easy to see that there are enough events according to β to complete the required task in every phase. By this and by the fact that there is no writing into the input variables, the correctness of the algorithm is easily deduced. We defer a detailed proof to the full paper.

For simplicity of demonstration, we have chosen a very simple algorithm to compute the majority of the inputs (in phase 0) and it may be improved. For example, according to [1], the majority can be computed without BS. Alternatively, an efficient algorithm $TTFM$ [6], for gathering the inputs at BS, can be used. In both alternatives, the algorithms can be transformed to their self-stabilizing versions by the transformer in [8] and then, used here in phase 0.

The multi-phase design using a phase clock may be useful in many other applications of mobile sensor networks. For instance, consider a habitat monitoring application such as ZebraNet [23] where sensors are attached to zebras. In this network, one may want to compute the highest temperature measured at a zebra (task T_1). Then, once this temperature t is computed, one may want to compute the percentage of females in the subset of zebras with a temperature near enough to t (task T_2). In a similar way, some additional sequential tasks could be programmed. The phase clock may be useful in detecting the termination of each task and then, in starting the execution of the next task in the sequence.

[6] Let us ignore the exact types of the input/output variables and the exact rules of the majority calculation, in this demonstration.

Group Mutual Exclusion (GME)

This problem has been introduced by Joung [22]. The problem deals with sharing μ mutually exclusive resources between $\mathbf{n}$ processors (agents, in our case). At any time, some agents share one particular resource. But, if some agent x requests to access a resource different from the currently used one, then x cannot access the requested resource at that time. However, if x requests the currently used resource, it is allowed to share the resource. There is no limit on the number of agents that can share the same resource. In addition, a fairness is required for accessing some resource and the waiting time (number of events, in our case) for a given resource is bounded. GME generalizes several synchronization problems such as mutual exclusion [15], dining philosophers [16], drinking philosophers [13], and more. The problem has been broadly studied in classical shared-memory and message-passing networks (see, e.g., [12], for a detailed survey) and also in ad hoc mobile networks (see, e.g., [21]). A self-stabilizing version of GME for asynchronous message-passing rings has been studied in [12]. In [12], the solution is token based and requires one distinguishable processor.

In the population protocols model we consider, designing a self-stabilizing solution for GME wouldn't be so easy. However, if we use the phase clock algorithm in Fig. 1, we almost automatically get a solution using a simple round robin strategy. Let $\mathbf{K} = 2\mu$ be the bound for the clock. The solution is the following: an agent x asking to access a resource r, waits for its clock to be equal to $2r$ and only then, accesses the resource, but releases it when its clock is equal to $2r + 1$. The correctness of the solution is implied by the correctness of the phase clock algorithm. When stabilized, the clocks of agents can differ by no more than 1. Thus, in such a state, at any moment, if an agent uses resource r, its clock value has to be $2r$ and at the same moment, no agent can have a clock value other than $2r$, $2r+1$ or $2r-1$. Hence, if at some moment, some agent uses resource r, no agent is allowed to use a resource other than r. In this solution, the *response time*, which is the maximum number of events since an agent asks for some resource and till it accesses the resource, is $O(\sum_{r=0}^{\mathbf{K}-1} \beta(r))$.

Note that in the proposed solution, the maximum ensured "time" that an agent may spend using resource r is $[\beta(2r)]^{BS}$ events (defined by the designer a priori), when in the original GME this time is assumed to be finite but unbounded. However, for some applications it may be enough for any agent preempted from the usage of a resource r to resume the usage at the next phase $2r$. An algorithm that solves GME with the original assumption (and still uses the presented phase clock) should be somewhat intricate.

Several applications of GME concerning the classical communication models can be found in [22]. An example is a CD jukebox (documentation, movies, etc.) on Internet: several users wishing to reach some CD will be able to do it at the same time instead of awaiting the end of the services of the other users.

In the model of population protocols, one could imagine still other applications. For instance, assume that each category of agents has its own frequency for emitting and each agent has to tune the frequency for receiving. Then, the clock would indicate what is the current frequency to receive or when agents

```
$\texttt{done}_x \in \{0,1\}$   /* indicates if a transition of task $\mathcal{A}$ has been executed by agent $x$ */

1 if ( $\texttt{clock}_x = \texttt{clock}_y \wedge (\texttt{clock}_x \mod 2 = 1)$ ) then   $\texttt{done}_x = \texttt{done}_y = 0$
2 if ( $\texttt{clock}_x = \texttt{clock}_y \wedge (\texttt{clock}_x \mod 2 = 0) \wedge \texttt{done}_x = \texttt{done}_y = 0$ ) then
3         $\langle$execute transition $(x,y)$ in $\mathcal{A}\rangle$
4         $\texttt{done}_x := \texttt{done}_y := 1$
```

Fig. 2. Synchronizer algorithm for **any** agent x in an event (x, y) with agent y

of the same category can communicate without interference (using predefined transmission/receiving frequencies).

Synchronizer

Many tasks in the model of mobile agents would be made easier if the agents could use perfect local clocks which could deliver fully synchronized successive ticks. At each tick all the agents would perform meetings, *simultaneously* and *pairwise*[7], and change their states accordingly to the transition function. These pairwise transitions are completed by the next tick. Unfortunately, such clocks are unrealistic, especially in a model where agents have very few resources. Fortunately, the ticks of such clocks can be simulated, by a so called synchronizer [4]. A synchronizer is a software, similar to a compiler, that takes as an input a protocol valid in the synchronous tick model and that outputs an asynchronous protocol for the same specification. In Fig. 2, we propose such a synchronizer that uses the phase clock. The synchronizer only reads the clock values provided by the phase clock. We assume that $\mathbf{K} > 2$ and is even. For every even p, $\beta(p) = \mathbf{cv_{max}}$. Otherwise, $\beta(p) = \mathbf{cv_{min}}$. The input protocol is denoted by $\mathcal{A}$ and its transitions are executed at line 3. Each agent x has a bit $\texttt{done}_x$ that indicates if a transition of $\mathcal{A}$ has been executed already by x during the current simulated tick. During every odd phase, `done` bits are reset and during every even phase, the synchronous tick with the transitions of $\mathcal{A}$ are simulated.

For the sake of place, the proof of correctness of the synchronizer is deferred to the complete paper. Below, we only show some main points of this proof.

After stabilization of the phase clock, by the *Frequency of progress* condition, every agent stays in an odd (even) phase (with the same odd (even) clock value) during at least $\mathbf{cv_{min}}$ ($\mathbf{cv_{max}}$) events.

By Lem. 7, after stabilization of the phase clock, whenever the first event of a new even (odd) phase occurs, it is ensured that during the previous $\mathbf{cv_{min}}$ ($\mathbf{cv_{max}}$) events, *all* the agents had odd (even) clock value. Whenever all the agents are in an odd phase during $\mathbf{cv_{min}}$ events, a fastest agent meets every other agent and they set their `done` bits to 0 (line 1). Thus, before the first event of a new even phase occurs, all the `done` bits are 0. Also, no agent that is in an even phase can reset its `done` bit to 0 (lines 1-4). Whenever all the agents are in

[7] We assume populations of even size. Recall that we deal with networks of *large* population of *limited and weak* sensors. Generally, applications in such networks would not relay on the correct operation of one or several agents from the population. Thus, even if $\mathbf{n}$ is odd, most of the applications would not be harmed if one agent did not perform a transition during a tick.

an even phase during $\mathbf{cv_{max}}$ events, every agent that has not yet executed line 3 (a transition of $\mathcal{A}$) during this phase, has exactly one opportunity (recall that we assume that $\mathbf{n}$ is even) to meet with another agent that has not yet executed line 3 either, during the current phase. When they meet, they both execute the required transition. Then, this scenario is repeated. During an odd phase, the agents reset their `done` bits. During an even phase, each agent executes exactly one transition of $\mathcal{A}$ (with an agent that did not execute yet any transition of $\mathcal{A}$, during the phase). Thus, after the stabilization of the phase clock, every couple of odd and then even phases, exactly simulates the execution of $\mathcal{A}$ during one tick in the synchronous model. In addition, all the events simulating tick $t + 1$ happen after all the events simulating tick t.

We conclude by giving an example of the usage of the presented synchronizer. Assume a mobile sensor application where agents have to perform some task $\mathcal{A}$ *repeatedly*. It may be the case that the operations related to this task are expensive in terms of, e.g., power consumption. In addition, the operations of $\mathcal{A}$ are to be executed only during a pairwise event (meeting) and using resources *of both* agents in the event. The condition on each repetition of $\mathcal{A}$ is that every agent performs the operation at least once. However, as the system is asynchronous, if not taking care, some fast agents may execute the expensive operations a larger than necessary number of times. The problem is to minimize the number of such operations. This problem is easily solved in the synchronous tick model presented above. Then, the synchronous solution should be given as an input to the synchronizer to obtain the asynchronous solution.

Remark 3. Note that in some cases, it may be useful to use the synchronizer in combination with the transformer of [8] to get a self-stabilizing protocol automatically. In such cases, a developer is proposed to design a protocol with initialization, in the synchronous tick model as above. Then, as the transformer is correct (in particular) in the synchronous model, it is applied to this protocol. Finally, the resulting self-stabilizing synchronous protocol is combined with the synchronizer to get a self-stabilizing asynchronous version of the protocol.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: PODC, pp. 290–299 (2004)
2. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. DC 21(3), 183–199 (2008)
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. DC 20(4), 279–304 (2007)
4. Awerbuch, B.: Complexity of network synchronization. Journal of the Association of the Computing Machinery 32(4), 804–823 (1985)
5. Awerbuch, B., Kutten, S., Mansour, Y., Patt-Shamir, B., Varghese, G.: A time-optimal self-stabilizing synchronizer using a phase clock. IEEE TDSC 4(3), 180–190 (2007)
6. Beauquier, J., Burman, J., Clement, J., Kutten, S.: Brief announcement: Non-self-stabilizing and self-stabilizing gathering in networks of mobile agents - the notion of speed. In: PODC, pp. 286–287 (2009)

7. Beauquier, J., Burman, J., Clement, J., Kutten, S.: On utilizing speed in networks of mobile agents. To appear in PODC 2010 (2010)
8. Beauquier, J., Burman, J., Kutten, S.: Making population protocols self-stabilizing. In: SSS, pp. 90–104 (2009)
9. Beauquier, J., Burman, J., Kutten, S.: A self-stabilizing transformer for population protocols with covering. Technical report, Technion (2010), http://tx.technion.ac.il/~bjanna/main_ss-transformer_els.pdf
10. Beauquier, J., Clement, J., Messika, S., Rosaz, L., Rozoy, B.: Self-stabilizing counting in mobile sensor networks with a base station. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 63–76. Springer, Heidelberg (2007)
11. Boulinier, C., Petit, F., Villain, V.: When graph theory helps self-stabilization. In: PODC, pp. 150–159 (2004)
12. Cantarell, S., Petit, F.: Self-stabilizing group mutual exclusion for asynchronous rings. In: OPODIS, pp. 71–90 (2000)
13. Chandy, K.M., Misra, J.: The drinking philosopher's problem. ACM Trans. Program. Lang. Syst. 6(4), 632–646 (1984)
14. Couvreur, J.-M., Francez, N., Gouda, M.G.: Asynchronous unison (extended abstract). In: ICDCS, pp. 486–493 (1992)
15. Dijkstra, E.W.: Solution of a problem in concurrent programming control. Commun. ACM 8(9), 569 (1965)
16. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta. Inf. 1, 115–138 (1971)
17. Fischer, M., Jiang, H.: Self-stabilizing leader election in networks of finite-state anonymous agents. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 395–409. Springer, Heidelberg (2006)
18. Guerraoui, R., Ruppert, E.: Even small birds are unique: Population protocols with identifiers. Technical Report CSE-2007-04. York University (2007)
19. Herman, T.: Phase clocks for transient fault repair. IEEE Trans. Parallel Distrib. Syst. 11(10), 1048–1057 (2000)
20. Herman, T., Ghosh, S.: Stabilizing phase-clocks. Inf. Process. Lett. 54(5), 259–265 (1995)
21. Jiang, J.-R.: A group mutual exclusion algorithm for ad hoc mobile networks. In: JCIS, pp. 266–270 (2002)
22. Joung, Y.-J.: Asynchronous group mutual exclusion. Distributed Computing 13(4), 189–206 (2000)
23. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., Rubenstein, D.: Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In: ASPLOS, pp. 96–107 (2002)
24. Lahde, S., Doering, M., Pöttner, W., Lammert, G., Wolf, L.C.: A practical analysis of communication characteristics for mobile and distributed pollution measurements on the road. Wirel. Comm. and Mob. Comput. 7(10), 1209–1218 (2007)
25. Lundquist, J.D., Cayan, D.R., Dettinger, M.D.: Meteorology and hydrology in Yosemite National Park: A sensor network application. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 518–528. Springer, Heidelberg (2003)
26. Massey, H., Bedford, L.H.: The role of satellites in observing and forecasting the global behaviour of the atmosphere: Discussion. Royal Society of London Proceedings Series A 308, 172 (1969)
27. Tel, G.: Introduction to Distributed Algorithms, 2nd edn. Cambridge University Press, Cambridge (2000)

Appendix

Proof of Theorem 1

Assume by contradiction that there exists a bounded generic solution G for a phase clock. By Def. 2, for an infinite set of populations of agents (for every $\mathbf{n}$ and for every vector of cover times) G provides an infinite set of bounded transition systems. From this set of systems, we extract an infinite sequence $\bar{S} \equiv S_1, S_2, \ldots$ with the following properties: (1) for any system S_i from $\bar{S}$, $\mathbf{cv_{min}}(S_i) \geq \frac{\mathbf{n}_{S_i} \cdot (\mathbf{n}_{S_i} - 1)}{2} + \mathbf{n}_{S_i}$, where $\mathbf{n}_{S_i}$ is the number of agents in S_i and $\mathbf{cv_{min}}(S_i)$ is the minimum cover time in system S_i; (2) for any two systems S_i and S_j (from $\bar{S}$), such that $i < j$, $\mathbf{cv_{min}}$ in S_j is greater than $\mathbf{cv_{max}}$ in S_i.

By the pigeonhole principle and because the systems in $\bar{S}$ are bounded, we can extract from $\bar{S}$ an infinite sub-sequence $\bar{S^*} \equiv S_1^*, S_2^*, \ldots$ such that for any two systems S_i^* and S_j^*, if $i < j$, the transitions of the agents in S_i^* and S_j^* are the same. Note that since every system in $\bar{S^*}$ is bounded, it can neither use $\mathbf{n}$, nor the values of the cover times.

Let us consider an execution e in a system S_i^* from $\bar{S^*}$. By the *Progress* condition of a phase clock (Def. 1), there is an agent x whose clock is incremented infinitely often in e. Then, assume $e = e_0 t_0 e_1 t_1 e_2 t_2 e'$, where the transitions t_0, t_1, t_2 cause consecutive increments of the clock of x. Then, by the *Asynchronous unison* condition of a phase clock and since $\mathbf{K} \geq 6$, in $t_0 e_1 t_1 e_2 t_2$, the clocks of all the other agents (of S_i^*) are incremented at least once (since the clock of x has been incremented three times).

Let $e_{pref} = e_0 t_0 e_1 t_1 e_2 t_2$. Let us choose from $\bar{S^*}$ a system S_j^*, such that $i < j$, $\mathbf{n}_{S_j^*} \geq |e_{pref}|$ and $\mathbf{n}_{S_j^*} > \mathbf{n}_{S_i^*}$. First, note that, because S_j^* is also in $\bar{S}$, e_{pref} satisfies the cover time property of S_j^* (since $\mathbf{cv_{min}}$ in S_j^* is greater than $\mathbf{cv_{max}}$ in S_i^*, implying that all the cover times in S_j^* are greater than the cover times in S_i^*). Second, recall that $\mathbf{cv_{min}}(S_j^*) \geq \frac{\mathbf{n}_{S_j^*} \cdot (\mathbf{n}_{S_j^*} - 1)}{2} + \mathbf{n}_{S_j^*}$ (because S_j^* is also in $\bar{S}$). Hence, $\mathbf{cv_{min}}(S_j^*) - |e_{pref}| \geq \frac{\mathbf{n}_{S_j^*} \cdot (\mathbf{n}_{S_j^*} - 1)}{2}$. Because during $\frac{\mathbf{n}_{S_j^*} \cdot (\mathbf{n}_{S_j^*} - 1)}{2}$ events every agent can meet every other agent, there is enough events ("time") after e_{pref} (in S_j^*) to satisfy $\mathbf{cv_{min}}(S_j^*)$ and every other $\mathbf{cv}$ of S_j^*. Thus, there exists an infinite schedule in S_j^* where e_{pref} is a prefix and the cover time property of S_j^* is satisfied in this schedule. E.g., one can get such a schedule by completing e_{pref} to get a prefix p of length $\mathbf{cv_{min}}(S_j^*)$ such that in p, every agent meets every other agent at least once. Then, repeat p indefinitely. Hence and because the transitions of agents in S_i^* and S_j^* are the same, e_{pref} is a possible prefix of an execution in S_j^*. However, because the population size in S_j^* is strictly greater than in S_i^*, there exists an agent y of S_j^* that does not participate in any event in e_{pref} and, in particular, in $t_0 e_1 t_1 e_2 t_2$. Thus, the clock of y has not been incremented there. Hence, the *Asynchronous unison* condition of a phase clock is not satisfied in the configuration at the end of e_{pref}, in S_j^*. This is a contradiction to the assumption that S_j^* is provided by a bounded generic solution for a phase clock. ∎

Sensor Allocation in Diverse Environments

Amotz Bar-Noy, Theodore Brown, and Simon Shamoun

Department of Computer Science, CUNY Graduate Center
amotz@sci.brooklyn.cuny.edu, tbrown@gc.cuny.edu, srshamoun@yahoo.com

Abstract. Sensor coverage varies with location due to factors such as weather, terrain, and obstacles. If a field can be partitioned into zones of homogeneous sensing areas, then coverage by a random deployment of sensors can be optimized by controlling the number of sensors deployed in each zone. We derive expressions to directly calculate the optimal sensor partition in runtime asymptotically equal to the number of zones. We further derive expressions to determine the minimum sensor count required to achieve a specific coverage threshold. We bound the maximum increase in coverage over a strategy oblivious to differences in sensing areas, which our results show is no greater than 13% for a field with two zones. While the analytical solutions assume that each zone is covered independently, we allow sensors to affect neighboring zones in simulations. Nevertheless, the simulations support the optimality of our solution.

Keywords: Sensor networks, coverage, deployment, optimization.

1 Introduction

A variety of environmental conditions may affect sensor coverage, such as terrain, weather, and vegetation. In many applications of sensor networks, these conditions vary throughout the field of deployment. For example, a sensor network for habitat monitoring in Southern California covers forests, brush, and meadows [12]. In a proposal to monitor amphibian populations in northern Australian with acoustic sensors, the region was divided into 2000 zones categorized by their levels vegetation and water resources [11]. As these conditions vary, so do the sensing areas of sensors. For example, an initial deployment of sensors to detect chemical plumes revealed irregularities in sensing ranges due to variations in terrain and meteorological conditions [16]. In order to maximize the coverage of a diverse environment, sensors must be distributed according to the coverage they provide at each location.

Several works propose deterministic deployment strategies for sensor networks in diverse environments [3,17,16,13,14]. They make few or no assumptions about how sensor coverage varies throughout the field of deployment, such that there may be no continuity in the change in sensing areas between neighboring points. The resulting algorithms are computationally complex and little or no analysis of quality of coverage is provided. These techniques can be simplified by realistically assuming that the field can be divided into zones of similar conditions, like in the Australian case, such that sensing area of a sensor is fairly uniform within each zone. This way, the sensors can first be allocated to each zone and then deployed within each zone according to well studied strategies for homogeneous fields.

This paper analyzes the problem of allocating sensors to uniform zones in a diverse field to maximize area coverage. The analysis relies on several broad assumptions: 1) The field can be divided into large enough zones such that common deployment strategies can be applied in each zone. For example, a grid layout is only applicable in a tall and wide region. 2) The deployment in each zone does not affect the deployment in any other zone. 3) The coverage of each zone can be calculated independently of all others. Although sensors affect coverage in bordering zones, simulations show that this is a reasonable assumption when zones are relatively large. Because of this assumption, the results are also applicable when zones do not border each other. The analysis is specifically for random deployment in each zone under further assumptions about the node distribution and models of coverage. While better coverage is possible by deterministic deployment, random deployment by methods such as airdropping is sometimes preferable and even necessary. Remote deployment causes fewer disturbances to natural habitats than direct placement. It is more expedient in urgent situations such as rescue operations. Direct placement is infeasible for large scale networks like the one considered for northern Australia, and is nearly impossible in hostile situations, such as battlefield operations. Nevertheless, the results of this paper can be extended to deterministic deployment strategies.

The paper begins with a review in Sect. 2 of three issues relevant to the analysis and simulation design: sensing models, models of environmental effects on coverage, and deployment strategies for diverse environments. The assumptions, notation, and formulas used in the analysis are defined in Sect. 3. Section 4 analyzes the problem for a field with two zones, presenting formulas for the optimal allocation, the minimum sensor count required for any expected level of coverage, and an upper bound on the maximum improvement in coverage over a baseline strategy that is oblivious to the difference in coverage between zones. These last two formulas are important in evaluating the expected costs and benefits of implementing such a strategy in place of the baseline strategy. It is shown that no more than 13% additional area is covered using the optimal strategy. Section 5 lists algorithms to derive the optimal allocation in fields with multiple zones in decreasing order of computational complexity, beginning with brute force and ending with direct computation. Although decreased complexity is preferable, the robustness of these algorithms decreases with complexity, as illustrated in Sect. 6. This section shows how to extend these algorithms to other coverage models and deployment strategies. Section 7 presents the results of simulations designed to test the assumptions upon which the analysis is constructed. While simulated coverage differs slightly from expected coverage, borders have minimal effect on the optimal partition. Finally, further problems for analysis are discussed in Sect. 8.

2 Related Research

The binary sensing model is the simplest and most widely analyzed model of coverage provided by a sensor. In this model, a sensor detects all events in its sensing area. Brass [1] gives upper bounds on the capabilities of random and deterministic strategies in bounded and unbounded regions. Liu and Towsley [8] analyze different measures of coverage provided by random deployment in large unbounded regions. These two works assume uniformly sized disk shaped sensing areas for all sensors and location assignment according to a Poisson point distribution in random deployments. Lazos and Poovendran [7] provide a more robust analysis of coverage by random deployment. Their analysis accounts for border effects in bounded regions and the size and shape of each sensor. They provide a limited analysis of coverage by non-uniform distributions. Lan, et al. [6] investigate the minimum sensing radius required to asymptotically achieve full coverage of the unit square by uniform and non-uniform random distributions. This is the inverse of the problem analyzed here, in that the distribution determines the sensor radius. For non-uniform distributions, they propose partitioning the square such that the distribution is uniform in each partition and setting the radii in each partition accordingly.

The probabilistic sensing model [3,17] models detection failure by assigning a detection probability to each point in a sensor's sensing area. Unlike in the binary model, increasing the number of sensors covering a point and their proximity to it increases detection probability. There are few analytical results for this model, such as optimal arrangements and expected coverage by random distributions. The general sensing model [8,15] is better studied. This model considers the aggregation of signals, subject to attenuation and noise, received by sensors. This model is only applicable, however, to network architectures and protocols that support this type of data aggregation.

Several papers [3,17] model environmental effects on coverage by representing the sensing area at each potential sensor location with a grid. Yang, et al. [16], correlate models of gas dispersion to actual terrain and weather data to derive the sensing areas of chemical sensors, resulting in ellipses of varying size. Fanimokun and Frolik [4] use experimental data to model the effects of three natural environments on the propagation of wireless transmissions. While these results are not for the propagation of sensing signals received by sensors, the same principles apply [8]. In this case, sensing ranges can be modeled with weighted distance functions [2].

Several works employ greedy algorithms to determine optimal sensor locations, modeling the sensing areas and deployment field with grids [3,17,16]. The complexity of these algorithms is $O(N^2m)$, in which N is the number of grid points in the field and m is the number of sensors deployed, which is very costly for fine grids. Yang, et al. [16] give an approximation factor of $\log C$ for this type of algorithm, in which C is the coverage requirement. Another approach [13] is to place sensors along a standard grid and adjust to the distances between grid points to improve coverage. This works well for uniformly shaped sensors, but is not very effective for highly diverse environments. A novel approach to distributing sensors [14] represents the field using a gray-scale image, in which the intensity varies with the sensing range, and then employs dithering algorithms to determine sensor locations in the field. No analysis is provided, though, of the resulting quality of coverage.

3 Preliminaries

The analysis here is for area coverage under the binary sensing model by the random uniform distribution of sensors in each zone. No assumptions are made about the shapes of sensing areas. The only assumption is that the sensing areas have the same area within each zone. Since the actual assignment of sensor locations is random, only *expected coverage* can be measured. Expected coverage is simply referred to as *coverage*, and the complement of coverage is referred to as *exposure*. Coverage and exposure are defined as the *fraction* of the area covered and not covered, respectively, by a sensor network. An *allocation* is the partition of sensors to be distributed in the different zones. In the *oblivious allocation*, the fraction of sensors allocated to a zone equals the fraction of the field the zone covers. Since this is the most reasonable strategy if no information about sensing areas is available, it is used as a base for comparing other strategies. *Absolute improvement* is the difference in coverage between two allocations. *Relative improvement* is the ratio of coverage by two different allocations. This paper strictly addresses absolute improvement from the oblivious allocation, which seems to be a more effective measure to discuss optimal random allocation while comparing it with other allocation solutions.

All measurements are made in arbitrary units. The total area of the sensor field is A, the maximum sensing area of any sensor is S, and the total number of sensors deployed is n. $\lambda = n/A$ is the *sensor density*, the average number of sensors per unit area. V refers to the fractional area coverage. The field is partitioned into m zones $(Z_1, Z_2, \ldots, Z_m)$. In general, for some portion of the field, whether a zone or a combination of zones, γA, $0 < \gamma < 1$, is its area, αS, $0 < \alpha < 1$, is the sensing area within it, and βn, $0 \leq \beta \leq 1$, is the number of sensors allocated to it. β can be a constant or a function of the other parameters. If $\beta_i n$ sensors are allocated to a zone with area $\gamma_i A$, then $\beta_i = \gamma_i$ is the oblivious allocation.

Liu and Towsley [8] cite a result in stochastic geometry [5] that the expected coverage of the infinite plane is $1 - e^{-S\lambda}$, where S is the expected sensing area. This formula can be modified to calculate the coverage of a bounded region with area A by n sensors *located within the field* [7]:

$$1 - e^{-\frac{Sn}{A}} \tag{1}$$

Lazos and Poovendran [7] derive a formula for coverage by n *heterogeneous* sensors that *intersect a field*, in which F_0 is the field area, L_0 is the field perimeter, and F_i and L_i are the area and perimeter, respectively, of each sensor i:

$$1 - \prod_{i=1}^{n} \left(\frac{2\pi F_0 + L_0 L_i}{2\pi(F_0 + F_i) + L_0 L_i} \right) \tag{2}$$

While (2) accounts for border effects and sensing area shapes more accurately than (1), equation (1) is still a fair estimate of coverage by homogeneous sensors [7], regardless of their size and shape. Additionally, independent tests conducted for this paper, the details of which are omitted, show that (1) is within 2% of the average coverage. Therefore, (1) is the foundation for all solutions in this paper. Based on (1), (3) can be used to determine the number of sensors required for expected coverage V.

$$n = -\ln(1-V)\frac{A}{S} \tag{3}$$

In the complexity analysis below, it is assumed that each arithmetic operation can be performed in constant time.

4 Two Zones

This section addresses sensor allocation in a field with only two zones, Z_1 and Z_2. The size of zones Z_1 and Z_2 are $(1-\gamma)A$ and γA, the sensing areas in each zone are S and αS, and number of sensors per zone are $n_1 = (1-\beta)n$ and $n_2 = \beta n$, respectively. See Table 1 for a summary.

Table 1. Parameters for two zones

Zone	Zone area	Sensing area	Sensor count
Z_1	$(1-\gamma)A$	S	$n_1 = (1-\beta)n$
Z_2	γA	αS	$n_2 = \beta n$

Based on (1), coverage can be calculated in $O(1)$ operations with the following formula:

$$\begin{aligned}&(1-\gamma)(1-e^{-\frac{S}{(1-\gamma)A}(1-\beta)n}) + \gamma(1-e^{-\frac{\alpha S}{\gamma A}\beta n})\\ &= 1-(1-\gamma)e^{-\frac{S}{(1-\gamma)A}(1-\beta)n} - \gamma e^{-\frac{\alpha S}{\gamma A}\beta n}\end{aligned} \tag{4}$$

This reduces to (1) when $\beta = \gamma$ and $\alpha = 1$. Two ways to derive the optimal allocation follow.

Solution 1 (Brute force, $O(n)$). Select the best of all $n+1$ possible partitions.

Solution 2 (Direct calculation, $O(1)$). Calculate n_2 by first calculating β_{opt} with (5) and then rounding $\beta_{opt} n$ to the integer for which coverage is maximum.

$$\beta_{opt} = \begin{cases} 0 & \text{if } n < -\frac{A(1-\gamma)\ln\alpha}{S} \\ \frac{1}{(1-\gamma)\alpha+\gamma}\left(\frac{A\gamma(1-\gamma)\ln\alpha}{Sn} + \gamma\right) & \text{otherwise} \end{cases} \tag{5}$$

Proof. Taking the second derivative of (4) shows that it is concave in the interval $[0,1]$. To find the value of β that maximizes (4), set the first derivative of (4) equal to 0 and solve for β. For values of n for which the resulting formula is less than 0, set β_{opt} equal to 0. $\square$

The next solution is used in estimating costs.

Solution 3. The number of sensors n required to achieve coverage V under the optimal partition can be calculated as follows:

$$n = \begin{cases} -\frac{(1-\gamma)A}{S} \ln \frac{1-\gamma-V}{1-\gamma} & \text{if } V < (1-\gamma)(1-\alpha) \\ -\frac{((1-\gamma)\alpha+\gamma)A}{\alpha S} \left(\frac{\alpha(1-\gamma)}{(1-\gamma)\alpha+\gamma} \ln \alpha + \ln \frac{1-V}{(1-\gamma)\alpha+\gamma} \right) & \text{otherwise} \end{cases} \tag{6}$$

Proof. First derive the following formula for V by substituting (5) for β in (4) and then solve for n.

$$V = \begin{cases} 1 - \left((1-\gamma) e^{-\frac{S}{(1-\gamma)A} n} + \gamma \right) & \text{if } n < -\frac{A(1-\gamma)\ln \alpha}{S} \\ 1 - ((1-\gamma)\alpha + \gamma) e^{-\frac{\alpha(1-\gamma)\ln\alpha}{(1-\gamma)\alpha+\gamma}} e^{-\frac{\alpha}{(1-\gamma)\alpha+\gamma}\frac{S}{A} n} & \text{otherwise} \end{cases} \tag{7}$$

□

The last solution is for bounding improvement over the oblivious allocation, which is to set $\beta = \gamma$. The upper bound depends on the value of β_{opt}. When $\beta_{opt} < \gamma$, the maximum improvement is trivially .5 when $S \to A$, $\gamma \to 0$ and $\alpha \to 0$ and only one sensor is deployed. In less extreme cases, significant coverage is only achieved when the sensor count is large enough such that $\beta_{opt} > \gamma$. Therefore, the focus here is on the case when $\beta_{opt} > \gamma$.

Solution 4. The upper bound in improvement when $\beta_{opt} > \gamma$, for any set of values of α, γ, n, A, and S, can be calculated with the following formula.

$$\gamma((1-\gamma)\alpha+\gamma)^{\frac{(1-\gamma)\alpha+\gamma}{(1-\gamma)(1-\alpha)}} \alpha^{\frac{\alpha}{1-\alpha}} (1-\alpha)(1-\gamma) \tag{8}$$

Proof. First, observe that β increases with n, since $\ln \alpha$ is negative for $\alpha < 1$ and $\lim_{n\to\infty} \frac{A\gamma(1-\gamma)\ln\alpha}{Sn} = 0$. Therefore, the improvement in Z_2 is an upper bound on the total improvement. Begin with the difference in coverage of Z_2 between using $\beta = \beta_{opt}$ and $\beta = \gamma$.

$$\gamma\left(e^{-\frac{\alpha S}{\gamma A}\gamma n} - e^{-\frac{\alpha S}{\gamma A}\frac{1}{(1-\gamma)\alpha+\gamma}\left(\frac{A\gamma(1-\gamma)\ln\alpha}{Sn}+\gamma\right)n}\right) \tag{9}$$

Find the value of n at which this is maximum by setting the derivative with respect to n equal to 0 and solving for n.

$$n = -\frac{A}{\alpha S}\left(\frac{(1-\gamma)\alpha+\gamma}{(1-\gamma)(1-\alpha)} \ln((1-\gamma)\alpha+\gamma) + \frac{\alpha}{1-\alpha}\ln\alpha \right) \tag{10}$$

Substitute this value of n back into (9) and reduce to find the upper bound. □

Remark 1. When $\beta_{opt} > \gamma$, absolute improvement never exceeds .13. The maximum value of (8) occurs when $\alpha \to 0$ and $\gamma \approx .394$, as found by numerical approximation.

5 Multiple Zones

Solutions for a field with m zones are discussed in this section. The area of each zone Z_i is $\gamma_i A$ such that $\sum_{i=1}^{m} \gamma_i = 1$. The sensing area in each Z_i is $\alpha_i S$, such that $1 = \alpha_1 > \alpha_2 > \ldots > \alpha_m > 0$. The number of sensors assigned to each Z_i is $\beta_i n$, such that $\sum_{i=1}^{m} \beta_i = 1$. The expected coverage is

$$\sum_{i=1}^{m} \gamma_i \left(1 - e^{-\frac{\alpha_i S}{\gamma_i A}\beta_i n}\right) = 1 - \sum_{i=1}^{m} \gamma_i e^{-\frac{\alpha_i S}{\gamma_i A}\beta_i n} \tag{11}$$

This is a generalization of (4), in which $\beta_1 = 1 - \beta_2$ and $\gamma_1 = 1 - \gamma_2$. The coverage of each zone can be calculated in $O(1)$ time and of the whole field in $O(m)$ time. Four ways to derive the optimal partition follow.

Solution 1 (Brute force, $O(n^{m-1}m)$). Select the best of all $\binom{n+m-1}{m-1}$ possible partitions.

Note 1. This is not considered polynomial in n, since, for $n = m$, $\binom{2n-1}{n-1} \geq \frac{2^{2n-1}}{\sqrt{n}}$.

Solution 2 (Dynamic programming, $O(n^2 m)$). Derive the optimal partition with the following dynamic programming formulation, in which $H(i, j)$ is the maximum achievable coverage of zones Z_i to Z_m with j sensors, and $G_i(j)$ is the coverage of Z_i by j sensors.

$$\begin{aligned} H(i,j) &= \max_{0 \leq k \leq j} \{\gamma_i G_i(k) + H(i+1, j-k)\} \\ H(m,j) &= \gamma_m G_m(j) \\ G_i(j) &= 1 - e^{-\frac{\alpha_i S}{\gamma_i A} j} \end{aligned} \tag{12}$$

Solution 3 (Greedy algorithm, $O(n \log m)$). Store the zones in a heap, in which priority is given to zones with the maximum increase in coverage when adding a sensor. Allocate sensors one by one by the following steps: Remove the top zone from the heap, add a sensor to it, calculate the coverage increase by adding another sensor to that zone, return the zone to the heap, and repeat.

Proof. A proof by contradiction shows that the greedy algorithm obtains the optimal partition. If the final partition is not optimal, then coverage can be improved by moving at least one sensor from Z_i to Z_j, for some i and j. Let this be the last sensor allocated to Z_i in the one-by-one allocation. The increase in coverage was greater by assigning it to Z_i rather than Z_j in the round that it was allocated to Z_i. The increase in coverage by any sensor added to Z_j afterwards must have been less than the increase by adding this sensor to Z_i. Otherwise, it would have been assigned to Z_j before them. Therefore, coverage cannot be improved by moving a sensor between zones. This is even true for moving several sensors, since the coverage increase declines with each sensor added to Z_j, while the coverage decrease grows as those same sensors are removed from Z_i.

The complexity is derived as follows. There are $O(n)$ rounds, one for each sensor. Each round requires $O(\log m)$ steps to reorder the heap. The coverage increase can be calculated in $O(1)$ time by applying (1). $\square$

Just as in the case of two zones, the runtime can be improved by calculating the optimal partition directly. The optimal β_i can be derived in $O(m^2)$ time with Lagrange multipliers, the details of which are excluded. An alternative solution is based on the following property of the coverage of two zones.

Proposition 1. *The optimal coverage of a two-zone field is equal to the expression* $1 - Ce^{-\frac{\alpha' S}{A}n}$ *for some values of* C *and* α'.

Proof. This is evident from (7). □

The direct calculation solution is for the following characterization of the field: Let Z'_i be the combination of zones $(Z_1, \ldots, Z_i)$. The area of Z'_i is $A'_i = \sum_{j=1}^{i} \gamma_j A$. Each zone Z'_i, for $i > 1$, can be considered a field of two zones Z'_{i-1} and Z_i. The fraction of Z'_i covered by Z_i is $\gamma'_i = \gamma_i / \sum_{j=1}^{i} \gamma_j$. Let β'_i be a partition of sensors between Z'_{i-1} and Z_i. See Table 2 for a summary of the parameters. The expected coverage is

$$1 - \sum_{i=1}^{m} \gamma_i e^{-\frac{\alpha_i S}{\gamma_i A}\beta'_i \prod_{j=i+1}^{m}(1-\beta'_j)n} \tag{13}$$

Table 2. Parameters for m zones

Zone	Zone area	sensing area	Sensor count
Z_1	$\gamma_1 A$	S	$(1-\beta'_2)\cdot \ldots \cdot (1-\beta'_m)n$
		$\ldots$	
Z_{m-1}	$\gamma_{m-1} A$	$\alpha_{m-1} S$	$\beta'_{m-1}(1-\beta'_m)n$
Z_m	$\gamma_m A$	$\alpha_m S$	$\beta'_m n$

Solution 4 (Direct calculation, $O(m)$). Begin by calculating β'_{opt_m} with (14) below, rounding $\beta'_{opt_m} n$ to the integer for which coverage is largest. Allocate these sensors to Z_m, and then recursively partition the remaining sensors in Z'_{m-1} using the same method, calculating $\beta'_{opt_{m-1}}$ for the remaining number of sensors.

$$\beta'_{opt_i} = \begin{cases} 0 & \text{if } n < -\dfrac{(1-\gamma'_i)A'_i \ln \frac{\alpha_i}{\alpha'_{i-1}C_{i-1}}}{\alpha'_{i-1}S} \\ \frac{1}{(1-\gamma'_i)\alpha_i + \gamma'_i\alpha'_{i-1}} \left(\dfrac{A\gamma_i(1-\gamma'_i)\ln \frac{\alpha_i}{\alpha'_{i-1}C_{i-1}}}{Sn} + \gamma'_i\alpha'_{i-1} \right) & \text{otherwise} \end{cases} \tag{14}$$

in which $A'_i = \sum_{j=1}^{i} \gamma_j A$, $\gamma'_i = \gamma_i / \sum_{j=1}^{i} \gamma_j$, and

$$C_1 = 1 \qquad C_{i\geq 2} = \frac{(1-\gamma'_i)\alpha_i + \gamma'_i\alpha'_{i-1}}{\alpha'_{i-1}} \left(\frac{\alpha'_{i-1}C_{i-1}}{\alpha_i} \right)^{\frac{(1-\gamma'_i)\alpha_i}{(1-\gamma'_i)\alpha_i + \gamma'_i\alpha'_{i-1}}} \tag{15}$$

$$\alpha'_1 = 1 \qquad \alpha'_{i\geq 2} = \frac{\alpha_i \alpha'_{i-1}}{(1-\gamma'_i)\alpha_i + \gamma'_i\alpha'_{i-1}} \tag{16}$$

Proof. According to Proposition 1, the optimal coverage of any zone Z'_i by n sensors is $1 - C_i e^{-\frac{\alpha'_i S}{A'_i} n}$, such that the expected coverage under a partition β'_i is formulated as

$$1 - (1 - \gamma'_i) C_{i-1} e^{-\frac{\alpha'_{i-1} S}{(1-\gamma'_i) A'_i}(1-\beta'_i) n} - \gamma'_i e^{-\frac{\alpha_i S}{\gamma'_i A'_i} \beta'_i n} \tag{17}$$

Taking the second derivative of (17) shows that it is concave. Therefore, there is one maximum for any value of β'_i. A sketch of the derivation of β'_{opt_i} is as follows. Set the first derivative of (17) with respect to β'_i equal to 0 and solve for β'_i.

$$\beta'_i = \frac{1}{(1-\gamma'_i)\alpha_i + \gamma'_i \alpha'_{i-1}} \left(\frac{A \gamma_i (1-\gamma'_i) \ln \frac{\alpha_i}{\alpha'_{i-1} C_{i-1}}}{Sn} + \gamma'_i \alpha'_{i-1} \right)$$

Substitute this expression into (17) and reduce to find the expression for coverage. This is used to define C_i and α'_i.

$$1 - \frac{(1 - \gamma'_i)\alpha_i + \gamma'_i \alpha'_{i-1}}{\alpha'_{i-1}} \left(\frac{\alpha'_{i-1} C_{i-1}}{\alpha_i} \right)^{\frac{(1-\gamma'_i)\alpha_i}{\gamma'_i \alpha'_{i-1} + (1-\gamma'_i)\alpha_i}} e^{-\frac{\alpha_i \alpha'_{i-1}}{\gamma'_i \alpha'_{i-1} + (1-\gamma'_i)\alpha_i} \frac{S}{A'_i} n} \tag{18}$$

□

Note 2. When $m = 2$, this equation equals (5).

With (14), it is possible to determine the minimum sensor count required to achieve an expected level of coverage.

Solution 5. The number of sensors n required to achieve coverage V under the optimal partition can be calculated as follows:

$$n = -\frac{A'_i}{\alpha'_i S} \ln \frac{\sum_{j=1}^{i} \gamma_j - V}{C_i \sum_{j=1}^{i} \gamma_j} \ , \ i = \max_{i \geq 1} \left(V \geq \sum_{j=1}^{i-1} \gamma_j \frac{\alpha'_{i-1} - \alpha_i}{\alpha'_{i-1}} \right) \tag{19}$$

Proof. In the optimal sensor partition, some zone Z'_i is the combination of all zones to which sensors are allocated. The coverage of the entire field is the coverage of Z'_i times the fraction of the field covered by Z'_i. Sensors are first allocated to Z_i when $n = -\frac{(1-\gamma'_i) A'_i \ln \frac{\alpha_i}{\alpha'_{i-1} C_{i-1}}}{\alpha'_{i-1} S}$. The optimal coverage of the entire field is therefore

$$V = \sum_{j=1}^{i} \gamma_j \left(1 - C_i e^{-\alpha'_i \frac{S}{A'_i} n} \right) \ , \ i = \max_{i \geq 1} \left(n \geq -\frac{(1 - \gamma'_i) A'_i \ln \frac{\alpha_i}{\alpha'_{i-1} C_{i-1}}}{\alpha'_{i-1} S} \right) \tag{20}$$

Here, $\alpha'_0 = C_0 = 1$. It can be shown that coverage of the entire field when $n = -\frac{(1-\gamma'_i) A'_i \ln \frac{\alpha_i}{\alpha'_{i-1} C_{i-1}}}{\alpha'_{i-1} S}$ is

$$\sum_{j=1}^{i} \gamma_j \left(1 - C_i e^{\frac{\alpha'_i (1-\gamma'_i) \ln \frac{\alpha_i}{\alpha'_{i-1} C_{i-1}}}{\alpha'_{i-1}}} \right) = \sum_{j=1}^{i-1} \gamma_j \frac{\alpha'_{i-1} - \alpha_i}{\alpha'_{i-1}} \tag{21}$$

□

The oblivious allocation is to set $\beta_i = \gamma_i$ in (11) or to set $\beta'_i = \gamma'_i$ in (13). The upper bound on improvement is no greater than the upper bound for a two zone field with $\alpha = \alpha_m$.

6 Extensions

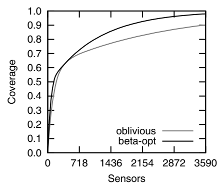
The solutions above can be modified to address many variations of the problem. For example, the greedy algorithm can be used if disk-shaped sensors are to be deployed deterministically. By making a loose assumption that sensors will be arranged along the grid points of a triangle lattice (which is the optimal arrangement of disks in an unbounded region [1]) in each zone, then a suitable equation for the coverage by such an arrangement of sensors (see [9] for an example) can be used to calculate the coverage increase by adding a sensor to each zone. Deployment should account for probable sensor failure [10]. If all sensors in Zone Z_i are associated with a probability of failure p_i, then Solution 4 can still be applied by replacing α_i with $p_i\alpha_i$ and ordering the zones accordingly. Many works consider regions of preferential coverage within the deployment field [3,17], such as densely populated areas [16] and amphibian hot-spots [11]. One way to address this is by weighting the coverage of each zone according to its priority, with the objective of maximizing weighted coverage [16]. In this case, (11) can be modified by multiplying the coverage of each zone by weight w_i.

In general, the allocation problem can be defined as follows: Given m functions $f_i(n_i)$ that characterize the quality of coverage of Z_i by n_i sensors, for all $1 \leq i \leq m$, and $g(f_1(n_1), \ldots, f_m(n_m))$ that characterizes the coverage of the an m-zone field by a partition $(n_1, n_2, \ldots, n_m)$ of n sensors, find a sensor partition that maximizes g. The scope of problems the solutions cover increases with their complexity. Direct calculation is only applicable for certain types of functions. The greedy algorithm applies when each f_i is strictly increasing, while dynamic programming is suitable for all types of functions f. The complexity of both these methods must be multiplied by the time to calculate f_i. Also, they are only suitable when g is linear. Brute force clearly covers all problems, but the complexity depends on the complexity of calculating coverage under each partition.

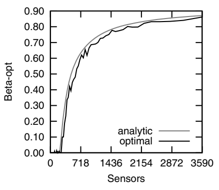
7 Simulation Results

In this section, simulation results are compared to expected values of (a) the coverage by the optimal and oblivious partition and the optimal deterministic deployment, (b) the maximum increase in coverage, and (c) the optimal beta for two zones. Coverage is approximated by superimposing a grid over the sensor field and counting the number of grid points that are covered by at least one sensor. All simulation results are averages of fifty or more repetitions.

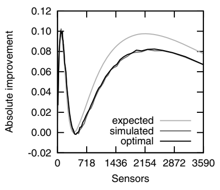
The first set of results are for square fields with two zones, in which all sensing areas are disks, $S = \pi 20^2$, and $A = 400^2$. Figure 1(a) compares the expected



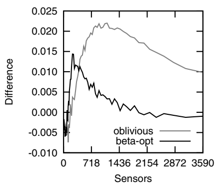
(a) coverage



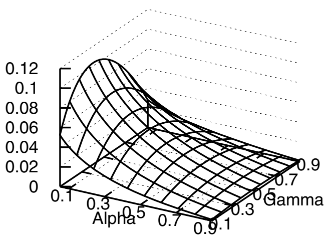
(b) partition



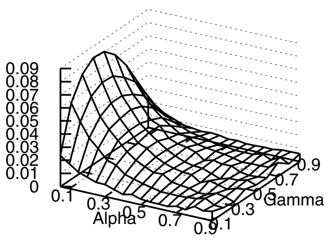
(c) improvement



(d) difference



(e) Upper bounds in improvement

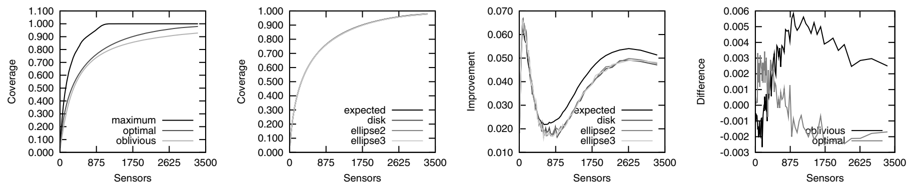


(f) Maximum improvement

Fig. 1. Coverage of two zones

coverage by the optimal partition to the expected coverage by the oblivious partition when $\alpha = .05$ and $\gamma = .4$. To evaluate the quality of Solution 2, β_{opt} is compared to the optimal partition found empirically using binary search in Fig. 1(b). The average improvement in coverage when using either partition is compared to the expected improvement when using β_{opt} in Fig. 1(c). It is clear from these figures that β_{opt} is indeed optimal. However, the average improvement is less than expected. Sensors in zone Z_1 significantly increase coverage in Z_2 when placed along the border between the two zones because of the large difference in sensing areas. This effect is not as great when using the optimal partition since the increase in coverage by adding sensors to either zone is the same. This effect is highlighted in Fig. 1(d), which shows the difference between the expected and average coverage by the oblivious and optimal partitions. The results are similar for other values of α and γ. Figure 1(e) shows the upper bounds in improvement according to (8) for $\alpha \in [.05, .90]$ and $\gamma \in [.1, .9]$, while Fig. 1(f) shows the maximum improvement found in simulations. The average improvement is smaller for the same considerations above.

The next set of results are for a square field with nine equal sized zones, in which $A = 900^2$ and $S_i \in \{\pi 8^2, \pi 12^2, \ldots, \pi 40^2\}$. Three sets of simulations were conducted, in which sensing areas in all zones were either disk shaped or ellipse-shaped with either a 2:1 or 3:1 major to minor axis ratio, denoted in the figures as "disk", "ellipse2", and "ellipse3", respectively. Additionally, these simulations were conducted for different arrangements of the zones in the field. Figure 2(a) compares an upper bound on the maximum achievable coverage by a deterministic partition, the expected coverage by the optimal partition,



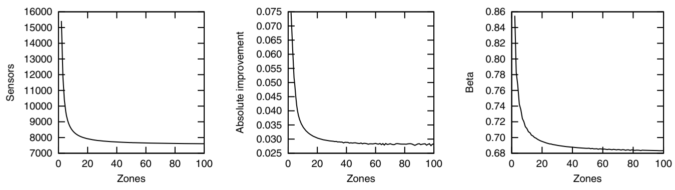
(a) expected (b) optimal (c) improvement (d) difference

Fig. 2. Coverage of a multi zone field

and the expected coverage by the oblivious partition. Figure 2(b) compares the expected coverage and average coverage by disk and ellipse shaped sensors under the optimal partition. In this instance, the difference between the expected and average coverage was less than 1% for all three shapes, providing some support to the no border effects assumption. In some other instances, the difference was close to 2%. However, coverage by disk and ellipse shaped sensors still differed by less than 1%, supporting the assumption the sensing shape has little effect on expected coverage. For the same consideration as in the two zone case, the average improvement is less than expected, as highlighted in Figures Fig. 2(c) and Fig. 2(d). The results were similar for fields with larger zones. However, when the sensing areas were increased, the differences in coverage between disk and ellipse shaped sensors became larger.

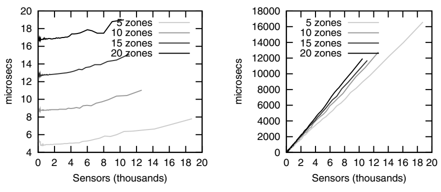
The following figures illustrate certain properties of the optimal partition in multi zone fields. In these figures, $A = 600^2$, $S = \pi 20^2$, $\gamma_i = 1/m$ for all i, $\alpha_m = .1$, and $\alpha_{i+1} - \alpha_i$ is the same for all i. Figure 3(a) shows the required number of sensors for an expected coverage of .98 by the optimal random allocation. Figure 3(b) shows the maximum improvement in coverage when all zones contain at least one sensor. Figure 3(c) shows the fraction of sensors allocated to the last $n/2$ zones when the expected optimal coverage is .98. These values quickly converge at about the same rate to specific values.

Because of the type and number of floating point operations required by the different solutions, the execution of the different algorithms was timed to verify the asymptotic runtime. Figure 4 shows the time required to calculate the optimal partition using the different algorithms. The values plotted are the medians



(a) sensor count (b) improvement (c) partition

Fig. 3. Statistics for multiple zone coverage



(a) direct (b) greedy

Fig. 4. Time to calculate optimal partition

of fifty simulations for each sensor count. The actual runtimes conform to the asymptotic runtimes, although the time for analytic solution slightly increases with the number of sensors.

8 Conclusion

This paper presents several solutions to the zone allocation problem for random deployment and shows how to extend these solutions to other problem models. Simulations supported the use of (1) in characterizing coverage and the assumptions that borders and shapes of sensing areas have little effect on the optimal partition. These assumptions are valid when the size of the sensing areas are not too large with respect to the size of the zones in which they are placed. Understanding the limitations of these assumptions requires further analysis, as well as a better understanding of the effects of borders and sensing shapes when these assumptions are no longer valid. Further analysis is required when the other assumptions upon which the analysis rests are loosened. For example, in reality, sensing areas are most likely somewhat diverse within each zone. Expected coverage under random deployments and optimal placement in "mildly" diverse zones must be addressed differently. The effects of inexact sensor deployment must be considered, both for deterministic [17] and random deployments (when sensors cannot be confined to the bounds of each zone). Finally, various other models can be considered, such as the general sensing model, non-uniform distributions, and weighted distance functions to represent environment effects.

Acknowledgments

This research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Brass, P.: Bounds on coverage and target detection capabilities for models of networks of mobile sensors. TOSN 3(2) (2007)
2. Cheung, Y.K., Daescu, O., Kurdia, A.: A new modeling for finding optimal weighted distances. In: Casadio, R., Daescu, O., Dini, C., Raicu, D.S. (eds.) BIOTECHNO, pp. 41–46. IEEE Computer Society, Los Alamitos (2008)
3. Dhillon, S.S., Chakrabarty, K.: Sensor placement for effective coverage and surveillance in distributed sensor networks. WCNC 3, 1609–1614 (2003)
4. Fanimokun, A., Frolik, J.: Effects of natural propagation environments on wireless sensor network coverage area. In: Proceedings of the 35th Southeastern Symposium on System Theory, pp. 16–20 (March 2003)
5. Hall, P.: Introduction to the Theory of Coverage Processes. John Wiley and Sons, Chichester (1988)
6. Lan, G.L., Ma, Z.M., Sun, S.S.: Coverage problem of wireless sensor networks. In: Akiyama, J., Chen, W.Y.C., Kano, M., Li, X., Yu, Q. (eds.) CJCDGCGT 2005. LNCS, vol. 4381, pp. 88–100. Springer, Heidelberg (2007)
7. Lazos, L., Poovendran, R.: Stochastic coverage in heterogeneous sensor networks. TOSN 2(3), 325–358 (2006)
8. Liu, B., Towsley, D.: A study of the coverage of large-scale sensor networks. In: MASS, pp. 475–483 (October 2004)
9. Pompili, D., Melodia, T., Akyildiz, I.F.: Three-dimensional and two-dimensional deployment analysis for underwater acoustic sensor networks. Ad Hoc Networks 7(4), 778–790 (2009)
10. Shakkottai, S., Srikant, R., Shroff, N.B.: Unreliable sensor grids: coverage, connectivity and diameter. Ad Hoc Networks 3(6), 702–716 (2005)
11. Shukla, S., Bulusu, N., Jha, S.: Cane-toad monitoring in kakadu national park using wireless sensor networks. In: Proceedings of APAN (2004)
12. Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A.M., Estrin, D.: Habitat monitoring with sensor networks. Commun. ACM 47(6), 34–40 (2004)
13. Verma, D.C., Wu, C.W., Brown, T., Bar-Noy, A., Shamoun, S., Nixon, M.: Location dependent heuristics for sensor coverage planning, vol. 6981, p. 69810. SPIE, San Jose (2008), http://link.aip.org/link/?PSI/6981/69810B/1
14. Verma, D.C., Wu, C.W., Brown, T., Bar-Noy, A., Shamoun, S., Nixon, M.: Application of halftoning algorithms to location dependent sensor placement. In: ISCAS, pp. 161–164 (2009)
15. Xing, G., Tan, R., Liu, B., Wang, J., Jia, X., Yi, C.W.: Data fusion improves the coverage of wireless sensor networks. In: Shin, K.G., Zhang, Y., Bagrodia, R., Govindan, R. (eds.) MOBICOM, pp. 157–168. ACM Press, New York (2009)
16. Yang, Y., Hou, I.H., Hou, J.C., Shankar, M., Rao, N.S.: Sensor placement revisited in a realistic environment. Tech. Rep. UIUCDCS-R-2007-2840, University of Illinois at Urbana-Champaign (2007)
17. Zou, Y., Chakrabarty, K.: Uncertainty-aware and coverage-oriented deployment for sensor networks. Journal of Parallel and Distributed Computing 64(7), 788–798 (2004)

Data Spider: A Resilient Mobile Basestation Protocol for Efficient Data Collection in Wireless Sensor Networks

Onur Soysal and Murat Demirbas

Computer Science & Engineering Dept.,
University at Buffalo, SUNY
{osoysal,demirbas}@cse.buffalo.edu

Abstract. Traditional deployments of wireless sensor networks (WSNs) rely on static basestations to collect data. For applications with highly spatio-temporal and dynamic data generation, such as tracking and detection applications, static basestations suffer from communication bottlenecks and long routes, which cause reliability and lifetime to plummet. To address this problem, we propose a holistic solution where the synergy of the WSN and the mobile basestation improves the reliability and lifetime of data collection. The WSN component of our solution is a lightweight dynamic routing tree maintenance protocol which tracks the location of the basestation to provide an always connected network. The mobile basestation component of our solution complements the dynamic tree reconfiguration protocol by trailing towards the data generation, and hence, reducing the number of hops the data needs to travel to the basestation. While both protocols are simple and lightweight, combined they lead to significant improvements in the reliability and lifetime of data collection. We provide an analytical discussion of our solution along with extensive simulations.

1 Introduction

The objective for deploying a wireless sensor network (WSN) is to collect data from an area for some time interval. Traditionally, a static basestation (SB) is deployed with the WSN, and the WSN nodes relay data over multihops towards the SB, which stores/uploads the data for processing. In order to improve the efficiency (which determines the lifetime) and reliability (which determines the quality) of data collection, most of the research in the literature focus on the relay nodes. Several schemes have been proposed for establishing coordinated sleep-wake-up, aggregation, and routing over relay nodes. On the other hand, relatively little attention is paid to changing the SB model, and investigating holistic solutions to the data collection problem.

The traditional SB model has several handicaps. A primary problem is that the SB constitutes a hotspot for the system. Since the nodes closer to SB are always employed in relaying the entire traffic, those nodes deplete their batteries quickly, putting a cap on the lifetime of the deployment. Another major problem is due to the spatio-temporal nature of the data generation. In several WSN deployments, including environmental monitoring [5], habitat monitoring [16], and especially surveillance systems [7,1], it has been observed that the phenomena of interest are local both in time and space. Fixing the location of the basestation ignores the nature of the data generation and results in long multihop paths for relaying, which leads to a lot of collisions and data losses.

In order to address the drawbacks of the SB model, several work proposed to deploy a mobile basestation (MB) for data collection [11,8,2]. The classical "data mule" work [13] proposed to exploit random movement of MBs to opportunistically collect data from a WSN. Here, the nodes buffer their data and upload only when the MB arrives within direct communication. Although this approach eliminates multihop data relaying, the tradeoff is the very high latency, which makes the approach unsuitable for real-time monitoring applications. To fix the latency problem, the mobile element scheduling (MES) work [15] considered the controlled mobility of the MB and studied the problem of planning a path for the MB to visit the nodes before their buffers overflow (which turned out to be an NP-complete problem [15,9]). MES work assumes that the datarates in the WSN are known and fixed (constant after initialization), and this is limiting for monitoring applications. Controlled sink mobility in [3] reduces latencies significantly through maintenance of routes to sink location from all nodes. Optimal solution for this model requires preprocessing similar to MES, so the authors in [3] propose a greedy alternative. However, since reactive sink mobility requires flooding of the entire network, the controlled sink mobility work [3] assumes that the sink stays for relatively long durations on small number of predefined sink locations, which limits its ability to address dynamic data generation in an agile manner.

In our previous work, we presented a holistic, network controlled MB algorithm, "data salmon" [6]. Data salmon constructs a backbone spanning tree over the WSN, and constrains both the data relaying and the MB movements to occur on this tree. Data salmon moves the MB greedily to the subtree where most of the traffic originates.[1] In return when the MB moves along one edge of the tree, it inverts the direction of the edge to point to its new location to ensure that the root of the backbone tree is switched to be at the new location of MB. Hence tracking of the MB is achieved with minimum cost. Although it achieves low cost tracking and reduces the average weighted relay distance of data, the data salmon has some shortcomings. The hotspot problem is not addressed: since data salmon uses a static backbone tree, the center of the static backbone tree still relays a significant amount of traffic and is a potential hotspot. Moreover, a static backbone tree implies that a message-loss during the handoff of MB from one node on the tree to the next leads to a permanent partitioning.[2]

[1] We showed that this greedy strategy is optimal, under the constraints of limiting all the data relaying to occur on the static backbone tree.

[2] Requiring acknowledgment messages alleviates the problem, but also increases the overhead of the protocol significantly.

Our contributions. We present **data spider** a holistic solution where the synergy of the WSN and the MB improves both the reliability and lifetime of data collection. The WSN component of our solution is a very lightweight dynamic routing tree maintenance protocol which tracks the MB to provide an always connected network. This tree is updated locally and efficiently by the movements of the MB. The visual imagery is that of a spider (corresponding to the MB) re-weaving/repairing its web (corresponding to the tree) as it moves. To complete the feedback loop, the spider relies on its web to detect interesting phenomena (data generation) to follow. By trailing towards the data generation, the MB component of our solution reduces the number of hops the data needs to travel to the basestation. Since the data spider uses a dynamically reconfigured tree to route traffic, it avoids the hotspot problem of data salmon, which used a static backbone tree for routing. As a result, data spider extends the lifetime of the deployment by several folds over the data salmon. Due to its dynamically reconfigured tree, data spider is also resilient against message-losses.

We present our dynamic tree reconfiguration protocol, DTR (read *detour*), in Section 3. The DTR philosophy is to update the tree at where it counts, that is, where the most recent action is. Therefore, instead of trying to maintain a distance-sensitive tree for the entire network (which is clearly a non-local task), we maintain a temporally-sensitive tree by reconfiguring the tree only at the immediate locality of the MB. Since we restrict the tree reconfiguration only to the singlehop of the MB, the maintenance cost of the tree is very low. Yet this does not lead to long and inefficient paths for data relaying to MB: since the MB follows the data generation closely, the effective length of data-forwarding paths is only a couple of hops. (The lenghts of non-data-forwarding paths are irrelevant for the data collection problem.) We give the correctness proof of tree reconfiguraton at DTR in Section 3.2.

To investigate the requirements for proper handoffs by the MB, we formulate the *handoff connectivity* property in Section 3.3. Handoff connectivity, intuitively, captures the notion of having no holes in the network. We note that our data spider waives the handoff connectivity requirement in practice. In Section 4 we present a simple yet very effective algorithm—trail-flow algorithm—for the MB, that avoids bad handoffs. In the trail-flow algorithm, the MB follows the edges where most data is flowing to itself, instead of going directly to the source of the data (which we dub as the follow-source approach). Our simulation results show that follow-source leads to several incorrect handoffs whereas trail-flow still functions correctly in the same density/network.

We give simulation results to investigate the scalability and efficiency of data spider, and compare it with data salmon and the SB approach in Section 5. Our simulator uses realistic lossy channel models and provides a high-fidelity energy calculation by using BMAC [12] as the model for the MAC layer communications.[3] Our simulation results show that data spider outperforms data salmon

[3] Since our simulator is parametrized extensively it is suitable for modeling and investigating other MB algorithms easily. Our simulator is available at `http://www.cse.buffalo.edu/ubicomp/dataSpider/`

and the SB approaches consistently, and leads to significant improvements in the reliability and lifetime of data collection. Although we focus on one mobile region of interest (ROI) and on one MB for most of the paper, we note that data spider extends readily to allow several MBs to share the same network to track multiple ROIs. Our simulation results with multiple MBs collecting data from multiple ROIs are very promising; despite the lack of explicit coordination between the MBs, these simulations show an emergent cooperation and division of labor among the MBs leading to improved performance.

2 Model

We consider a dense, connected, multihop WSN. The sensor nodes are static after the initial deployment. We assume that the data generation has spatio-temporal correlation but is otherwise dynamic/unpredictable. This model captures the data generation in event detection, tracking, and surveillance applications. Our implementation of the data generation uses a circular region of interest (ROI). We allow only the sensors in this ROI to generate data. The ROI moves around in the network nondeterministically to simulate the behavior of mobile events. This ROI scheme generates dynamic data that is challenging for precomputed basestation mobility as in [3,9].

We account for the message transmission costs of the sensor nodes as well as the reception costs and idle listening costs at the nodes. We assume CSMA with BMAC low-power-listening [12] for the MAC layer and use the associated energy model in [12] to calculate the energy usage at each node. Since our DTR and MB protocols are simple, we ignore the energy cost of the computation. We assume the MB is capable of locating itself and traveling to target locations. In our model, we have not included the energy required for moving the MB.[4]

3 Dynamic Tree Reconfiguration

Here, we first present the DTR algorithm. We give the correctness proof of DTR in Section 3.2 and present the handoff connectivity requirements for DTR in Section 3.3. Finally, we present extensions to the basic DTR in Section 3.4.

3.1 DTR Algorithm

To maintain always-on connectivity to the MB, the network should continuously track it and update the existing routing paths to point to its new location. Trying to maintain a distance-sensitive tracking structure (e.g., maintaining a shortest

[4] The reason behind our willingness to generously tradeoff the energy required for relocating the MB with the energy gain in data collection is that it is much easier to replenish and maintain the batteries of one MB than those of the sensor nodes in the entire network. We assume that the MB is recharged periodically or is equipped with energy harvesting capabilities such as solar panels.

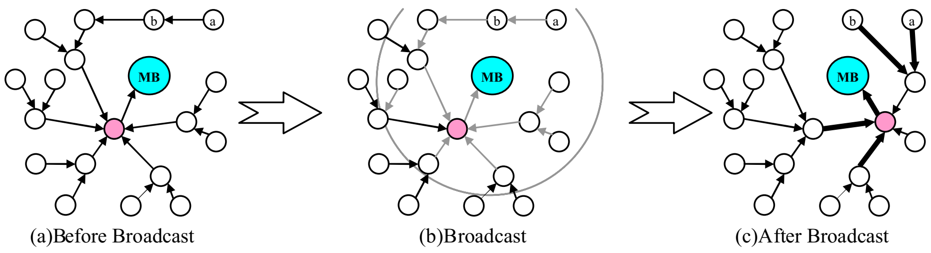


Fig. 1. Demonstration of DTR as MB moves from one anchor to another. The touched edges are gray edges in (b) and actual changes are bold edges in (c).

path tree rooted at the MB) would be beneficial since it would reduce the number of hops data need to be relayed towards the MB. However, this is inherently a non-local and costly task as it requires frequent multihop broadcasts.

Since energy-efficiency is of utmost importance for elongating the lifetime, in our dynamic tree configuration protocol, DTR, we take an alternative approach. To keep the maintenance cost of the tree very low, we confine DTR to reconfigure the tree only at the immediate locality (singlehop) of the MB. To ensure that DTR does not beget long and inefficient paths for data relaying to MB, we rely on the MB algorithm. In our simulation results in Section 5.2, we show that since the MB's trail-flow algorithm follows the data generation closely, the effective length of data relaying paths is only a couple of hops.

DTR starts with a spanning tree rooted at the MB. This could be established by constructing an initial tree using flooding and keeping the MB static. The root node of this initial tree is called the *anchor* node, which is also the closest node to the MB. As it relocates in the network, MB chooses the anchor node to be the closest node to itself and makes periodic broadcasts to declare the anchor node to all nodes in its singlehop range. Nodes that receive the anchor broadcast update their parents (next pointers) to point to the new anchor node. At any time there is a unique anchor node in the network, which is maintained to be the closest node to the MB.

We present DTR in Algorithm 1. Only the nodes that receive the anchor broadcast execute an action and update their next pointers. The anchor broadcasts are local to the singlehop of the MB and they are not relayed to multiple hops. Figure 1 depicts an example of DTR execution.

Algorithm 1. DTR Algorithm

```
1. Wait for the anchor message
2. **if** anchor == self **then**
3.    next ← MB
4. **else if** anchor ∈ Neighbors **then**
5.    next ← anchor
6. **end if**
```

Dynamic convoy tree work [17] adresses a relevant dynamic tree reconfiguration problem in the context of target tracking. Dynamic convoy tree maintains a monitoring tree to cover a mobile ROI. The root of this monitoring tree controls expansion and contraction of the tree and when needed decides on the relocation of the root to another node based on the information it collects from the entire tree. Our advantage in DTR is the cooperation of the MB for relocating the root of the tree to an optimal location using local singlehop updates, whereas the convoy tree needs to deal with the tree reconfiguration problem by using multihop update messages.

3.2 Correctness

We call the operation with which the MB changes the anchor node from one node to another as *handoff*. We call a handoff a *proper handoff* iff (1) both the old and new anchor nodes receive the MB's anchor broadcast, and (2) both the old and new anchor nodes can reliably communicate with each other. Finally, we call a network *routing connected* iff the *next* links of nodes form a spanning routing tree rooted at MB.

Theorem 1. *If all handoffs are proper, an iteration of Algorithm 1 starting from a routing connected network results in another routing connected network.*

Proof. Consider tree reconfiguration on a graph $G = (V, E)$ where $u, v \in V$ correspond to the nodes and $e = (u, v) \in E$ correspond to the reliable communication links between the nodes. We use r to denote the old anchor and r' to denote the new anchor. In the base case, when there is no handoff, $r' = r$, and the theorem holds vacuously. We next consider the case where $r' \neq r$.

The iteration of Algorithm 1 entails an anchor broadcast received by a set of nodes $R \subset V$. Let $S \subseteq R$ be the set of nodes that actually change their *next* links as a result of executing Algorithm 1. Since proper handoffs are assumed, $\{r, r'\} \subseteq S$. Algorithm 1 dictates that all nodes in S points to r' (with the exception of r' which points to the MB) after the update. That is, the *next* links of nodes in S form a routing tree rooted at r'.

Let $T(r)$ be a spanning routing tree of G rooted at node r, and F_S be the forest obtained by removing the *next* links of nodes in S from $T(r)$. Since $r \in S$, each tree in F_S is rooted at a node in $s \in S$. By definition, none of the edges in any tree $T_s \in F_S$ is changed. Since *next* links in S forms a routing tree rooted at r', *next* links in F_S and S form a spanning routing tree rooted at r'.

While message losses are common in WSN environments, most message losses do not affect the correctness of DTR (Theorem 1), as the definition of proper handoffs only require reliable message delivery between the MB and the old and new anchors. For the remaining nodes, message loss is only a nuisance, rather than constituting a correctness problem. Message losses at these nodes may result only in degraded performance, since their path is not updated to point to the new anchor in the most direct/shortest manner. But, since the previous routes point to the old anchor, which points to the new anchor, due to Theorem 1 the network is still routing-connected.

The routing-connected network property is violated only when the old or new anchor miss an anchor broadcast. DTR deals with this problem in two timescales: short and long terms. In the short term the impact of message losses are reduced through message redundancy. Increasing the anchor broadcast frequency at the MB improves the chances that all neighbors receive the information about the new anchor node. When this scheme is insufficient, there may be partitions in the network due to improper handoffs. In the long term, since the MB is mobile, MB is very likely to move over the partitioned regions eventually. This will, in turn, fix the problem and enable the buffered packets to be relayed to the MB.

3.3 Handoff Connectivity

The correctness of DTR depends on the success of handoffs, which is in turn imposed by the geometry and topology of the network. Here, we focus on planar deployments and capture these required geometric and topological properties.

In data spider, MB invariantly maintains its closest node as the anchor node. A useful abstraction for capturing this property is the Voronoi diagram of WSN nodes. When the MB is in one of the Voronoi cells, its closest node, by definition, is the WSN node corresponding to that Voronoi cell. Thus, as long as MB stays in that Voronoi cell, the anchor node is unchanged.

With this anchor node definition, we identify the requirements for having proper handoffs as follows. Let P denote a point in the deployment area and V_P be the set of nodes which are closest to P. So, if P falls inside a Voronoi cell, then V_P consists of a single node, the WSN node corresponding to that Voronoi cell. If P falls on a Voronoi cell boundary, then V_P consists of the neighboring (i.e., adjacent) nodes for this Voronoi cell boundary.

We call a WSN deployment **handoff connected** when all points P in deployment region satisfy:

1. For all nodes $n \in V_P$, n can reliably communicate with a node placed at P.
2. For all nodes $n, m \in V_P$, n and m can reliably communicate with each other.

In other words, in a handoff connected network (1) the MB sitting on a Voronoi cell boundary can communicate with the nodes in the adjacent Voronoi cells, and (2) any pair of Voronoi neighbors can communicate with each other.

The above handoff connectivity definition is valid when the updates of the MB are continuous. Since we use discrete/periodic anchor broadcasts, we extend this definition for our model. Let $\lambda = v_{BS} * T_{update}$ be the maximum distance the MB can travel between two location updates. We now require the anchor node to be able to receive messages from the MB when it is at most λ away from the Voronoi cell. Moreover, for proper handoff, any cell that falls to this region should be in communication range. Figure 2 demonstrates this requirement.

To generalize the handoff region we extend the set of nearest nodes V_P. V_P^λ to be the set of nodes which are at most $\lambda + d_{min}$ away from point P where d_{min} is the minimum distance to any node in network from P. Thus, using V_P^λ we generalize *handoff connectivity* as follows:

A WSN deployment is said to be **λ-general handoff connected** when all points P satisfy:

1. For all nodes $n \in V_P^\lambda$, n can reliably communicate with a node place at P.
2. For all nodes $n, m \in V_P^\lambda$, n and m can reliably communicate with each other.

3.4 Extensions to DTR

Handoff connectivity addresses only the immediate neighborhood of the anchor node. Broadcasts on the other hand can be made stronger with better transmitters on the MB so WSN nodes can receive broadcasts from non-neighboring Voronoi cells. These receptions can be utilized to improve the performance of DTR as follows. For this operation, nodes depend on neighborhood information about their neighbors. That is, nodes share neighborhood information with their neighbors, so that they can create two-hop routes to the anchor node when singlehop routes are not possible. If the anchor is not an immediate neighbor of the node, the node chooses its neighbor which is an immediate neighbor of the anchor. In case there are multiple neighbors satisfying this condition, the closest one to the anchor is chosen as the next node. As long as the chosen intermediate nodes also received the anchor broadcast this operation extends the handoff connectivity. We call this operation *indirect handoff*. We show neighbor nodes where only indirect handoff is possible with green(light) edges in Figure 2. Non-anchor nodes also benefit from our indirect handoff extension, as is the case for nodes a and b in Figure 1.

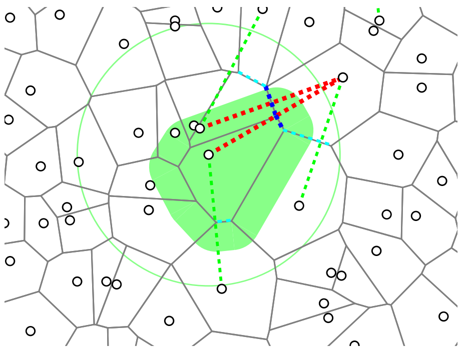


Fig. 2. Shaded region shows possible locations of MB at the next update, starting from center. Circle denotes the reliable communication range. Green(light) dashed lines correspond to neighbors where direct handoff is not possible. Red(dark) dashed lines correspond to neighbors where no handoff is possible.

4 MB Algorithm

The MB algorithm (given in Algorithm 2) synergizes with the DTR algorithm to achieve efficient data collection. MB ensures two things:

Algorithm 2. MB Algorithm

```
1. loop
2.    listen and update RecentPackets
3.    if count(RecentPackets) > 0 then
4.       target ← getTarget()
5.    else
6.       target ← getRandomTarget()
7.    end if
8.    navigate to target
9.    anchor ← closestNodeTo(position)
10.   broadcast anchor message
11. end loop
```

(1) The MB broadcasts an anchor message announcing the closest node to itself periodically. This enables DTR to track MB correctly and update the tree to be rooted at the MB. In order to detect and announce the closest node to itself, MB should know its location as well as the locations of nodes in the network. This is achievable by equipping the MB with a GPS and the coordinates of the WSN nodes. Having a GPS on the MB is relatively cheap, and the MB can also utilize its GPS to locate and collect the nodes after deployment.
(2) The MB relocates to follow data generation in a best-effort manner. This relocation reduces the length of data relaying paths in DTR to be a couple of hops, improving both the reliability and the lifetime. To track the data generation, MB utilizes the recent data packets that DTR routes to itself to decide where to move to next. MB defaults to a random walk when there are no packets, since this might indicate a disconnection of the network. Random walk may help the MB to repair the partitioning and re-establish a connected network where DTR can start delivering the data generated to the MB. Otherwise, MB uses the *getTarget()* function to decide how to relocate based on the recently received packets. We propose two heuristics for this function:

trailSource. Here, the MB inspects the source field of the data packets and sets the relocation target to be the source of the packet generation (median of the source locations). Although it seems like a reasonable approach, we show in Section 5 that when the network is not regular (has holes in it) trailSource leads to many improper handoffs and suffers severe performance penalties.

trailFlow. Here, the MB tries to go to the center of packet flow. In contrast to trailSource that calculates the center of data generation, trailFlow calculates the center of data forwarding from the singlehop neighbors of the MB. Since packet forwarding is done over reliable edges, trailFlow directs the MB to avoid the holes in the network implicitly (as a side benefit), so even in irregular and sparse networks trailFlow ensures successful handoffs. Our simulation results in Section 5 show that trailFlow consistently performs the best compared to the other heuristics.

5 Simulation

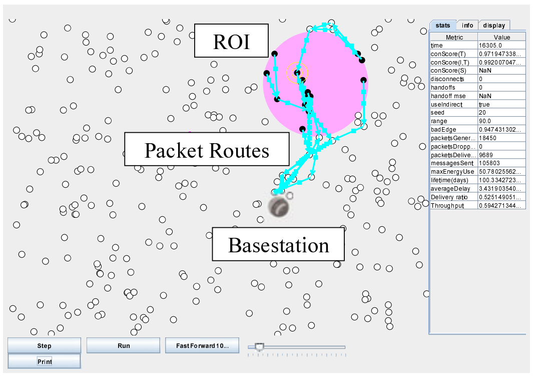
5.1 Setup

Simulator. We built our simulator on top of the JProwler simulator [14] and implemented support for mobility for JProwler. Our simulator is parametrized extensively, so it is suitable for modeling and investigating other MB algorithms quickly. Figure 3 shows a screenshot of our simulator. Our simulator and details about our simulator implementation are available at `http://www.cse.buffalo.edu/ubicomp/dataSpider/`.

Simulation setup. We set the simulation area as a 160m by 120m rectangular region with 300 mica2 nodes. We constrain the MB to this region and assume that there are no significant obstacles to obstruct mobility within the region. We model the data generation activity in the environment with a moving disc to denote the ROI. The region of interest is a circle with 20m radius. All WSN nodes covered by this disc generate data with a predetermined rate. The nodes then try to forward this data to the MB if they have a valid *next* link. A node buffers data if the channel is busy, or if it does not have a valid *next* link—which may happen after an improper handoff. We leverage on work in [10] to generate realistic human/animal like mobility patterns for the ROI.

In order to address energy efficiency questions we keep track of energy use in our simulation. Our simulator uses CSMA with BMAC low-power-listening [12] for the MAC layer and the associated energy model to calculate the energy used in each node. In our simulation we obtain fine grain information about packet arrivals and noise and replace the approximate values used in [12] with these values to better capture the energy use in each sensor node. Table 1 summarizes the parameters used for the energy efficiency calculations.

We ran each set of simulations for 72 simulation hours. Each simulation includes an initial neighborhood discovery and initial flooding phase.



Metric	Value
time	16305.0
conScore(T)	0.971947338...
conScore(I,T)	0.992007047...
conScore(S)	NaN
disconnects	0
handoffs	0
handoff mse	NaN
useIndirect	true
seed	20
range	90.0
badEdge	0.947431302...
packetsGener...	18450
packetsDropp...	0
packetsDelive...	9689
messagesSent	105803
maxEnergyUse	50.78025562...
lifetime(days)	100.3342723...
averageDelay	3.431903540...
Delivery ratio	0.525149051...
Throughput	0.594271344...

Fig. 3. A screenshot of the simulator. Dark circles are the data generating nodes, and shaded (pink) circle is the ROI. MB is indicated by the tiny roomba picture and arrows show the packet routes.

Table 1. Parameters used for energy efficiency calculations

Parameter	**Value**
Radio sampling interval	0.1s
Energy cost of a packet transmission	7.62mJ
Energy cost of a packet reception	3.18mJ
Energy cost of LPL for one second	0.263mJ
Battery Capacity	27000J

Neighborhood discovery phase reduces the disconnections and message losses as reliable links are identified and each node discovers its neighbors. This neighborhood information is later utilized in performing indirect handoffs. We **do include** the communication in this phase in our energy cost figures as well.

Protocols we compare with. We are primarily interested in evaluating the data spider system which consists of DTR and the MB algorithms, trailSource and trailFlow, described in Sections 3 and 4. For comparison, we also consider three other protocols, namely, *static*, *random*, and *salmon*.

In the static protocol, the basestation is static and is located in the center of the network. The data is routed to the based using a convergecast tree rooted at the SB. As we discussed in the Introduction this scheme is prone to hotspots around the SB, and also results in long multihop paths for data relaying.

The random protocol is similar to trailSource and trailFlow in that it also uses the DTR protocol to reconfigure the data collection tree as the MB relocates. However, as for the relocation algorithm, instead of trying to follow the data generation, the random protocol prescribes relocating the MB to a random location all the time. While this protocol avoids the hotspot issue (since it uses an MB and DTR), it is prone to long multihop paths for data relaying as it does not follow the data generation.

Salmon protocol uses the same MB algorithm we used in our previous work, data salmon [6]. Salmon does not use DTR and constrains the relocation of the MB to occur only along the edges of the existing tree. In other words, the existing tree is not modified, except for the relocation of the root of the tree from one node to one of the neighboring nodes (which is achieved by flipping the direction of the edge between these two nodes). In this scheme, the MB chooses the neighbor that forwards the majority of the traffic to relocate to. As our simulation results exhibit, this scheme has problems with reliability (since only one edge is modified, this constitutes a risk of single point of failure) and cannot follow the data generation successfully (since the MB relocation is restricted to the existing tree structure, MB needs take long detours when the ROI leaves the current subtree for another subtree).

Metrics. We concentrate on three metrics to measure performance of the system. The latency metric measures the average delay in packet deliveries, from their generation time to their arrival to MB. The second metric, packet delivery rate, is the ratio of the delivered packets to MB versus the number of packets generated. The final metric is the estimated lifetime of the network. We define the lifetime to be the time passed until the first node failure due to battery depletion in the network. By utilizing the fine-grained energy-use information from our simulation and the total energy stored in standard AA batteries, we arrive to our estimated lifetime figures.

5.2 Results

We present our simulation results under the following categories.

Node density. We first investigate the effect of node density on the performance. As the number of nodes increase, since the distance between anchor nodes would be decreasing, we expect better connectivity of the network and reduced number of improper handoffs. Increased density also corresponds to increased data rates and more contention reducing the lifetime of the network. Figure 4 presents this axis of the investigation. We observe very high latencies when node density is low. This is due to frequent disconnections. Packets are buffered when handoffs can not be completed successfully and they are later retrieved on an opportunistic basis, but this results in high average latencies. Data spider heuristics *trailFlow* and *trailSource* consistently outperform other protocols with respect to packet deliveries and network lifetime. An interesting result of this experiment is to show that even random mobility leads to better delivery ratios than the static when the density is critically low. Random mobility leads to worse delivery ratios when the density increases, yet it still leads to longer lifetimes than static (recall that random still uses DTR for data collection).

Indirect handoff. Here we quantify the performance improvement due the indirect handoff extension. Our experiments in Figure 5 show that indirect handoff provides better average latencies, and up-to 5% improvement in packet delivery rates. The increased number of packet deliveries impact the lifetime as more packets are successfully routed all the way to the sink.

Speed of region of interest (ROI). The ability to track ROI is a significant advantage for data spider, but the performance of tracking is affected by the speed of ROI. In our experiments depicted in Figure 6 we investigate the effect of speed of ROI to the performance. Since we use a fixed speed for MB, increasing

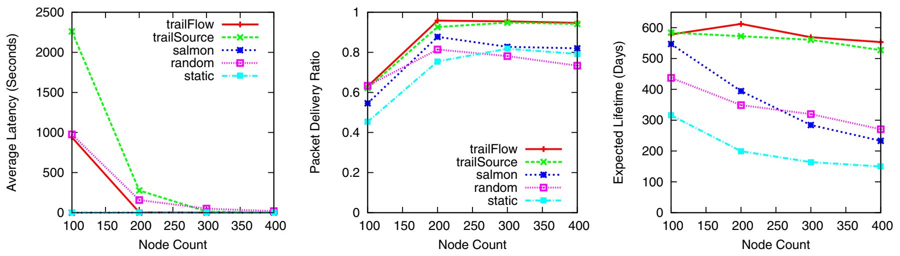


Fig. 4. Effect of number of nodes on performance

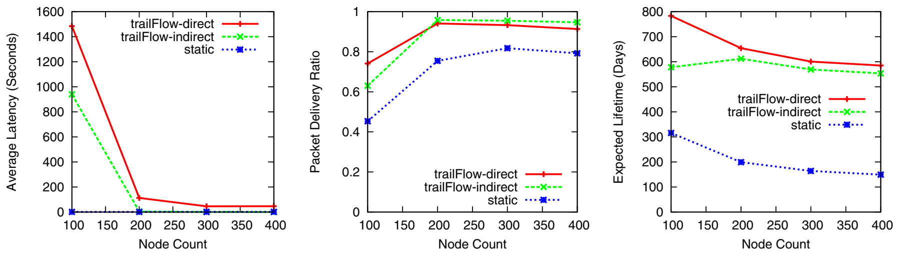


Fig. 5. Effect of indirect handoff on performance

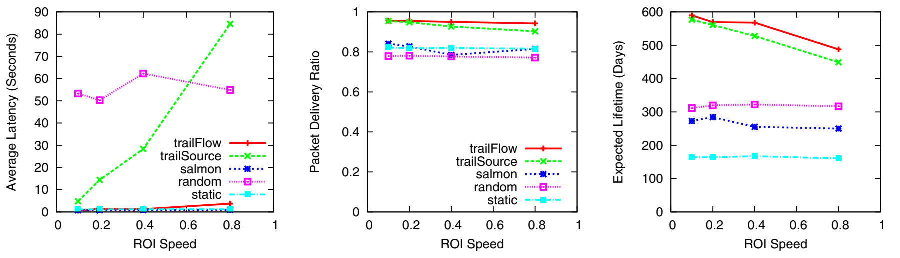


Fig. 6. Effect of speed of region of interest on performance

the speed of ROI makes tracking the data more difficult. As expected static and random heuristics are not effected by the ROI speed. We observe significant increase in average delay in *trailSource* heuristic. This increase is related to increased number of bad handoffs, which leads to partitions of network. *trailFlow* avoids this problem as packets follow the network topology and the MB follows the packets. Even with increased ROI speed, data spider algorithm improves the lifetime of network up to 3 times over SB.

Number of ROI. We next consider the effects of increasing number of ROIs on the performance. As these ROIs move independently from each other, the optimal location of MB would vary significantly and the static MB starts to become a better alternative. Our simulation results are shown in Figure 7. We observe the effect of disconnections in *trailSource* heuristic in this experiment as well. The difference between data delivery rates decrease as data spider heuristics can not follow all the ROIs at the same time. Lifetime of the network is also inversely affected as the MB is constrained to a smaller region trying to follow all ROIs simultaneously. With 4 ROIs, the performance of data spider is similar to random MB in terms of network lifetime, which is still more than 100% improvement over the SB.

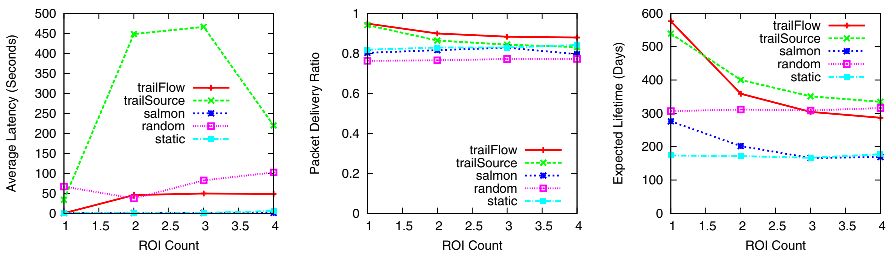


Fig. 7. Effect of number of region of interests on performance

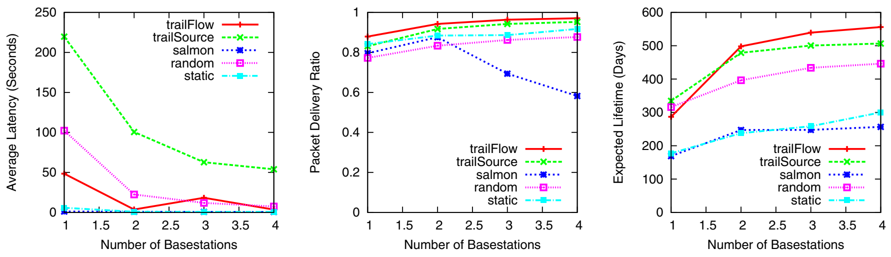


Fig. 8. Effect of number of MBs on performance with 4 independent ROIs

Multiple MBs. Figure 7 showed that increasing the number of ROIs reduced the ability of data spider to track them. Here we show how the increased number of ROIs are better handled with multiple MBs. We test the performance of data spider with multiple MBs in Figure 8. As we mentioned in the Introduction, data spider extends readily to allow multiple MBs to share the same network without any need to change the DTR or MB algorithms. In this experiment neither the network nor the MBs are aware of the multiple MBs. However, we still observe an emergent cooperation and division of labor leading to improved performance. MBs partition the network since each node only has one next node, moreover these partitions dynamically change over time due to MB broadcasts. Even if all MBs converge to same anchor, the competition for data allows MBs to diverge and cover different ROIs. We obtained these very promising results with data spider despite lack of explicit coordination. An interesting research question is how to coordinate MBs in a cooperative manner to improve performance even further. Recent studies focus on this direction [4].

6 Concluding Remarks

We presented an efficient holistic MB-based data collection system, data spider, which consists of a dynamic tree reconfiguration protocol and an MB protocol. While both protocols are simple and lightweight, combined they lead to significant improvements in the reliability and lifetime of data collection, especially for monitoring applications with highly spatiotemporal data generation. We provided extensive simulation results evaluating the latency, cost, and network lifetime metrics of the data spider under a wide number of varying parameters. We also analyzed the handoff connectivity requirements needed for performing a proper handoff of the MB.

Although we focused on the data collection problem, our data spider framework readily applies also to the pursuer-evader tracking problem by treating the ROI as the evader and the MB as the pursuer. Our experiments showed that the trail-flow algorithm for the MB manages to implicitly route the MB around the holes, a desirable property for pursuer-evader tracking. Our experiments also showed that, in the data spider system, multiple MBs coexist nicely on the same network to trail multiple ROIs without any explicit coordination or cooperation. In future work we will investigate coordination and cooperation mechanisms of multiple MBs for more efficient pursuer-evader tracking.

References

1. Arora, A., et al.: Exscal: Elements of an extreme scale wireless sensor network. In: RTCSA (2005)
2. Azad, A., Chockalingam, A.: Mobile base stations placement and energy aware routing in wireless sensor networks. In: WCNC, vol. 1, pp. 264–269 (April 2006)
3. Basagni, S., Carosi, A., Melachrinoudis, E., Petrioli, C., Wang, Z.M.: Controlled sink mobility for prolonging wireless sensor networks lifetime. Wirel. Netw. 14(6), 831–858 (2008)
4. Basagni, S., Carosi, A., Petrioli, C.: Heuristics for lifetime maximization in wireless sensor networks with multiple mobile sinks. In: IEEE International Conference on Communications, ICC 2009, June 2009, pp. 1–6 (2009)
5. Batalin, M., et al.: Call and response: experiments in sampling the environment. In: SenSys 2004, pp. 25–38 (2004)
6. Demirbas, M., Soysal, O., Tosun, A.S.: Data salmon: A greedy mobile basestation protocol for efficient data collection in wireless sensor networks. In: Aspnes, J., Scheideler, C., Arora, A., Madden, S. (eds.) DCOSS 2007. LNCS, vol. 4549, pp. 267–280. Springer, Heidelberg (2007)
7. Arora, A., et al.: A line in the sand: A wireless sensor network for target detection, classification and tracking. Computer Networks 46(5), 605–634 (2004)
8. Gandham, S., Dawande, M., Prakash, R., Venkatesan, S.: Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In: GLOBECOM, vol. 1, pp. 377–381 (December 2003)
9. Gu, Y., Bozdag, D., Ekici, E., Ozguner, F., Lee, C.: Partitioning based mobile element scheduling in wireless sensor networks. In: SECON, pp. 386–395 (2005)
10. jen Hsu, W., Spyropoulos, T., Psounis, K., Helmy, A.: Modeling time-variant user mobility in wireless mobile networks. In: INFOCOM, pp. 758–766 (2007)
11. Luo, J., Hubaux, J.-P.: Joint mobility and routing for lifetime elongation in wireless sensor networks. In: INFOCOM, vol. 3, pp. 1735–1746 (2005)

12. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys, pp. 95–107 (2004)
13. Shah, R.C., Roy, S., Jain, S., Brunette, W.: Data mules: modeling a three-tier architecture for sparse sensor networks. In: WSNPA, pp. 30–41 (2003)
14. Simon, G., Volgyesi, P., Maroti, M., Ledeczi, A.: Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In: IEEE Aerospace Conference, pp. 255–267 (March 2003)
15. Somasundara, A., Ramamoorthy, A., Srivastava, M.: Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In: RTSS, pp. 296–305 (2004)
16. Szewczyk, R., Mainwaring, A., Polastre, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: SenSys (2004)
17. Zhang, W., Cao, G.: Dctc: Dynamic convoy tree-based collaboration for target tracking in sensor networks. IEEE Transactions on Wireless Communication 3(5), 1689–1701 (2004)

Author Index