Utz Roedig
Cormac J. Sreenan (Eds.)

LNCS 5432

Wireless Sensor Networks

6th European Conference, EWSN 2009
Cork, Ireland, February 2009
Proceedings

Springer

Lecture Notes in Computer Science 5432

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Alfred Kobsa
University of California, Irvine, CA, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Utz Roedig Cormac J. Sreenan (Eds.)

Wireless Sensor Networks

6th European Conference, EWSN 2009
Cork, Ireland, February 11-13, 2009
Proceedings

Springer

Volume Editors

Utz Roedig
Lancaster University
InfoLab21
Lancaster, LA1 4WA, UK
E-mail: u.roedig@lancaster.ac.uk

Cormac J. Sreenan
University College Cork
Department of Computer Science
Kane Building, College Road, Cork, Ireland
E-mail: cjs@cs.ucc.ie

Library of Congress Control Number: 2008944097

CR Subject Classification (1998): C.2.4, C.2, F.2, D.1.3, D.2, E.1, H.4, C.3

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN 0302-9743
ISBN-10 3-642-00223-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-00223-6 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2009
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12613724 06/3180 5 4 3 2 1 0

Preface

This volume contains the proceedings of EWSN 2009, the 6th European Conference on Wireless Sensor Networks. The conference took place in Cork, Ireland during February 11–13, 2009. The aim of the conference was to discuss the latest research results and developments in the field of wireless sensor networks.

EWSN received a total of 145 full paper submissions of which 23 were selected for publication and presentation, yielding an acceptance rate of just under 16%. Paper submissions were received from 36 different countries in all parts of the world. EWSN adopted a double-blind review process, where the identities of the paper authors were also withheld from the reviewers. The selection process involved well over 400 reviews with all papers being evaluated by at least three independent reviewers. In addition, the reviews were discussed by the Technical Program Committee after collecting all reviews and prior to making final decisions. The final program covered a wide range of topics which were grouped into six sessions: performance and quality of service, routing, coordination and synchronization, data collection, security, evaluation and management. It included theoretical and analytical approaches, together with empirical research and protocol/system design and implementation.

The conference included a demo and poster session, co-chaired by Dirk Pesch and Sajal Das, for which separate proceedings are available. In addition, the conference included a keynote address by John Stankovic entitled "Wireless Sensor Networks: Time for Real-Time?", a panel discussion moderated by Jorge Pereira on the topic "What Is Holding WSNs Back? Breakthroughs or Mindsets?", a tutorial by Luca Mottola and Gian Pietro Picco on "Programming WSNs: From Theory to Practice," and a tutorial by Mario Alves on "WSN Standards and COTS Landscape: Can We Get QoS and Calm Technology?" The Conference was preceded by a workshop—the First European TinyOS Technology Exchange (ETTX 2009).

We would like to thank everyone who contributed to EWSN 2009. In particular, we would like to thank the Technical Program Committee for their reviews and input in forming the program. We also would like to thank the local administration at University College Cork for their help with the conference planning, and last but certainly not least our sponsors—Cork Institute of Technology (Platinum Sponsor), CONET Network of Excellence (Gold Sponsor), Tyndall National Institute (Gold Sponsor) and University College Cork (Silver Sponsor).

February 2009

Utz Roedig
Cormac J. Sreenan

Organization

EWSN 2009, the 6th European Conference on Wireless Sensor Networks, took place in Cork, Ireland, February 11–13, 2009. The conference was organized by Lancaster University (UK) and University College Cork (Ireland).

Conference and Technical Program Committee Co-chairs

Utz Roedig	Lancaster University, UK
Cormac J. Sreenan	University College Cork, Ireland

Posters and Demos Co-chairs

Dirk Pesch	Cork Institute of Technology, Ireland
Sajal K. Das	University Texas at Arlington, USA

Publicity Chairs

Chenyang Lu	Washington University in St. Louis, USA
Jorge Sa Silva	University of Coimbra, Portugal
Rosalind Wang	CSIRO, Australia

Program Committee

Tarek Abdelzaher	University of Illinois, Urbana Champaign
Habib M. Ammari	Hofstra University, New York
Michael Beigl	TU Braunschweig
Jan Beutel	ETH Zurich
Philippe Bonnet	University of Copenhagen
Athanassios Boulis	National ICT Australia
Torsten Braun	University of Bern
Nirupama Bulusu	Portland State University
Andrew Campbell	Dartmouth College
Srdjan Capkun	ETH Zurich
Mun Choon Chan	National University of Singapore
Peter Corke	CSIRO
Andrzej Duda	Grenoble Informatics Laboratory
Deborah Estrin	University of California at Los Angeles
Hannes Frey	University of Paderborn
Mike Hazas	Lancaster University
Sanjay Jha	University of NSW

Holger Karl	University of Paderborn
Ralph Kling	Crossbow
Srdjan Krco	Ericsson Ireland
Koen Langendoen	Delft University of Technology
Pedro Marron	University of Bonn and Fraunhofer IAIS
Suman Nath	Microsoft Research
Brendan O' Flynn	Tyndall National Institude
Joseph Paradiso	MIT
Joe Polastre	Sentilla
Nissanka Priyantha	Microsoft Research
Kay Roemer	ETH Zurich
Andreas Savvides	Yale University
John Stankovic	University of Virginia
Andreas Terzis	Johns Hopkins University
Roberto Verdone	University of Bologna
Thiemo Voigt	Swedish Institute of Computer Science
Dirk Westhoff	NEC Europe
Andreas Willig	Technical University of Berlin
Adam Wolisz	Technical University of Berlin

Additional Reviewers

Markus Anwander	Alexander Gluhak	Benoit Ponsard
Abdelmalik Bachir	Gertjan Halkes	Silvia Santini
Chiara Buratti	Muhammad Haroon	Robert Sauter
Shafique Chaudhry	Philipp Hurni	Chia-Yen Shih
Tony Chung	Umer Iqbal	Petcharat Suriyachai
Virgini Corvino	Venkat Iyer	Hwee-Xian Tan
Benoit Darties	Nicholas Lane	Shao Tao
Shane Eisenman	Hong Lu	Fabrice Theoleyre
Jia Fang	Andreas Meier	Nicolas Tsiftes
Szymon Fedor	Emiliano Miluzzo	Markus Waelchli
Matthias Gauger	Daniel Minder	Gerald Wagenknecht
Eugenio Giordano	Prasant Misra	Matthias Woehrle

Sponsors

Platinum: Cork Institute of Technology
Gold: CONET Network of Excellence, Tyndall National Institute
Silver: University College Cork

Table of Contents

Performance and Quality of Service

Routing

Coordination and Synchronization

Data Collection

Security

Evaluation and Management

Area Throughput of an IEEE 802.15.4 Based Wireless Sensor Network

Chiara Buratti, Flavio Fabbri, and Roberto Verdone

WiLab - DEIS at University of Bologna,
V.le Risorgimento, 2
I-40136 Bologna, Italy
{c.buratti,flavio.fabbri,roberto.verdone}@unibo.it

Abstract. In this paper we present a mathematical model to evaluate the performance of an IEEE 802.15.4 based multi-sink Wireless Sensor Network (WSN). Both sensors and sinks are assumed to be Poisson distributed in a given finite domain. Sinks send periodic and synchronous queries, and each sensor transmits its sample to a sink, selected among those that are audible. The IEEE 802.15.4 Multiple Access Control (MAC) protocol is used by sensors to access the channel: both Beacon-Enabled and Non Beacon-Enabled modes are considered. Our aim is to describe how the *Area Throughput*, defined as the amount of samples per unit of time successfully transmitted to the sinks from the given area, depends on the density of sensors and the query interval. We jointly account for MAC issues (i.e. packet collisions) and connectivity aspects (i.e. network topology, power losses and radio channel behaviour). Performance is evaluated by varying the traffic offered to the network (i.e. the number of sensors deployed), the packet size, the number of Guaranteed Time Slots allocated, and the Superframe Order. A comparison between results obtained in the Beacon-Enabled and the Non Beacon-Enabled cases is also provided, showing how the *Area Throughput* can be optimised.

1 Introduction

Many applications of Wireless Sensor Networks (WSNs) deal with the estimation of spatial/temporal random processes [1], [2]. Sensors are deployed in the target area, which is observed through query/respond mechanisms: queries are periodically generated by the application, and sensor nodes react by sampling and sending data to a centralised unit. By collecting samples taken from different locations and observing their temporal variations, estimates of the target random process realisation can be produced [3]. Good estimates require sufficient data taken from the area. Often, the data must be sampled from a specific portion of space, even if the sensor nodes are distributed over a larger area. Therefore, only a location-driven subset of sensor nodes must respond to queries. The aim of the query/response mechanism is then to acquire the largest possible number of samples from the area.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 1–16, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The data taken from the area are transmitted to the unit by means of wireless links connecting sensors to sinks, which collect the samples and forward them to the unit through a proper network. If few sensor nodes are deployed and the target area is small, a single sink can be used. When the number of sensors or the target area are large, they are often organised in clusters; one sink per cluster forwards the queries to sensors, and collects the responses.

Sinks are sometimes specifically deployed in optimised and planned locations with respect to sensors. However, opportunistic exploitation of the presence of sinks, connected to the centralised unit through some mobile radio interface, is an option in some cases [4]. Under these circumstances, many sinks can be present in the monitored space, but their positions are unknown and unplanned; therefore, achievement of a sufficient level of samples is not guaranteed, because the sensor nodes might not reach any sinks (and thus be isolated) due to the limited transmission range. In such an uncoordinated environment, network connectivity is a relevant issue, and it is basically dominated by the randomness of radio channel and the density of sinks.

Being the acquisition of samples from the target area the main issue for the application scenario considered, we define a new metric for studying the behavior of the WSN, namely the *Area Throughput*, denoting the amount of samples per unit of time successfully transmitted to the centralised unit, originating from the target area. We evaluate the *Area Throughput* in this paper, assuming that sinks forward the data collected to the centralized unit with no losses: therefore, only the WSN (sensors and sinks) is modelled. As expected, the *Area Throughput* is larger if the density of sensor nodes is larger, but, on the other hand, if a contention-based MAC protocol is used, the density of nodes significantly affects the ability of the protocol to avoid packet collisions (i.e. simultaneous transmissions from separate sensors towards the same sink). If, in fact, the number of sensor nodes per cluster is very large, collisions and backoff procedures can make data transmission impossible under time-constrained conditions, and the samples taken from sensors do not reach the sinks and, consequently, the centralised unit. Therefore, the optimisation of the *Area Throughput* requires proper dimensioning of the density of sensors, in a framework model where both MAC and connectivity issues are considered.

We assume sensors transmit their samples to sinks through an IEEE 802.15.4 air interface. IEEE 802.15.4 [5] is a short-range wireless technology intended to provide applications with relaxed throughput and latency requirements in wireless Personal Area Networks (PANs). In our scenario, sinks will act as PAN coordinators, periodically transmitting queries to sensors and waiting for replies. The 802.15.4 standard offers two types of channel access mechanisms: Beacon and Non Beacon-Enabled. The latter case uses unslotted CSMA/CA (Carrier Sensing Multiple Access with Collision Avoidance), whereas in the former operation mode a slotted CSMA/CA algorithm and a superframe structure managed by the PAN coordinator, is used. According to the standard, the different PAN coordinators, and therefore the PANs, use different frequency channels (see the scan functionality performed by the PAN coordinator for establishing a PAN).

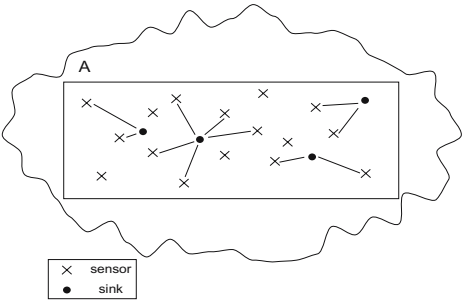


Fig. 1. The Reference Scenario considered

Therefore no collisions may occur between nodes belonging to different PANs; however, nodes belonging to the same cluster, will compete to try to transmit their packets to the sink.

We consider here an infinite area where sensors and sinks are uniformly distributed at random. Then, we define a specific portion of space, of finite size and given shape (without loss of generality, we consider a square), as the target area; both the number of sensors and sinks are then Poisson distributed in such area (see Figure 1).

We define as *Offered Load* the amount of samples available from the sensors deployed in the area, per unit time. The basic objective of this paper is thus to determine how the *Area Throughput* depends on the *Offered Load* and the parameters of the IEEE 802.15.4 MAC protocol used. Both MAC and connectivity issues are taken into account under a joint approach.

In general terms, we might say that we aim at defining a picture showing how throughput varies with load, as done for many years in the literature for different types of MAC protocols: here, we also include connectivity and the plurality of sinks into account. The aim of this paper is to compare the *Area Throughput* obtained in the Beacon-Enabled and Non Beacon-Enabled modes, as a function of all MAC parameters, the density of nodes and the query interval. The study is completely performed through mathematical analysis.

The rest of the paper is organised as follows. The following Section deals with related works, Section 3 introduces the scenario and the link model. In Section 4 the *Area Throughput* is evaluated, by computing the success probability for the transmission of a packet accounting for connectivity and MAC issues. In Section 5 the IEEE 802.15.4 MAC protocol model for the Beacon- and Non Beacon-Enabled cases is introduced, for the evaluation of the success probability related to MAC. Finally, in Section 6 and 7 numerical results and conclusions are presented.

2 Related Works

Many works in the literature are related to the modelling of different MAC protocols, and also to connectivity models, but very few papers jointly consider the two issues under a mathematical approach. Some analysis of the two issues

are performed through simulations: as examples, [6] related to ad hoc networks, and [7], to WSNs. Many papers devoted their attention to connectivity issues of wireless ad-hoc and sensor networks in the past (e.g., [8]). Single-sink scenarios have attracted more attention so far. However, an example of multi-sink scenario can be found in [9]. All the previously cited works do not account for MAC issues.

Concerning the analytical modelling of the IEEE 802.15.4 MAC protocol few works devoted their attention to Non Beacon-Enabled mode [10]; most of the analytical models are related to Beacon-Enabled networks [11,12,13,14]. In [10], the authors try to model the unslotted CSMA/CA protocol for non beacon-enabled networks, but they do not capture correctly the protocol, because they include in the Markov chain two subsequent sensing phases, and not one, as fixed in the Standard (see Section 5).

[11] fails to match simulation results [12], as the authors use the same Markov formulation and assumptions made by Bianchi in [15], where the 802.11 MAC protocol is considered. A better, even if similar, model is proposed in [13], where, however, the sensing states are not correctly captured by the Markov chain. In [12] the main gaps of the previous models are overcome. However, in all the previous works, the probability to find the channel busy is evaluated regardless of the backoff stage in which the node is. Our model, instead, captures the different probabilities at the different backoff stages. This leads to a better match between simulation and analytical model results (see [17,18,19]). Moreover, the aforementioned models assume that nodes always (or at least with a certain probability) have a packet to be transmitted. In this case, when a node succeeds in transmitting its packet, it will restart the backoff algorithm, possibly with a certain probability. In our model, instead, nodes transmit only one packet per received query, according to the applications considered. Finally, all the above cited works study the asymptotic behavior of the network, that is the behavior of the system at the equilibrium conditions, evaluating the stationary probabilities, obtained when time, t, approaches infinite ($t \to \infty$). Our analysis, instead, evaluates the statistical distribution of some metrics over time, starting from the reception of the query sent by the sink.

The model proposed here is based on the following previous works. In [16] the authors presented a mathematical model for the evaluation of the degree of connectivity of a multi-sink WSN in unbounded and bounded domains. [17, 18] provide a mathematical model to derive the success probability for the transmission of a packet in an IEEE 802.15.4 single-sink scenario when the Non Beacon-Enabled mode is used; in [19] the mathematical model for the IEEE 802.15.4 Beacon-Enabled mode, is provided. Finally, in [20] the concept of *Area Throughput* has been introduced, and the Non Beacon-Enabled mode has been considered as an example case. All these contributions are combined here to achieve the goal of this paper.

3 Assumptions and Reference Scenario

The reference scenario considered consists of an area of finite size and given shape, where sensors and sinks are both distributed according to a homogeneous

Poisson Point Process (PPP). We denote as $\rho_s[m^{-2}]$ and $\rho_0[m^{-2}]$ the sensors and sinks densities, respectively, and with A the area of the target domain. Denoting by k the number of sensor nodes in A, k is Poisson distributed with mean $\bar{k} = \rho_s \cdot A$ and p.d.f.

$$g_k = \frac{\bar{k}^k e^{-\bar{k}}}{k!}. \tag{1}$$

We also denote as $I = \rho_0 \cdot A$ the average number of sinks in A.

We assume that sinks periodically send queries to sensors and wait for replies. In case a sensor node receives a query from more than one sink, it selects the one providing the largest received power and responds to it. Upon reception of the query, each node will take one sample from the environment and will start the CSMA/CA-based algorithm (slotted and unslotted, depending on the modality selected), to try to transmit the data to the sink, through a direct link. We assume that each node transmits a packet having size $D{\cdot}10$ Bytes, with D being a positive integer. Since an IEEE 802.15.4 air interface is considered, the time needed to transmit a packet is equal to $D \cdot T$, where $T = 320\ \mu sec$, as a bit rate of 250 kbit/sec is used. Moreover, we set the size of the query equal to 60 Bytes.

We denote as $f_q = 1/T_q$ the frequency of the queries transmitted by the sinks, being T_q is the time interval between two consecutive queries.

3.1 The Link Model

The link model that we exploit accounts for the power loss due to propagation effects including both a distance-dependent path loss and the random channel fluctuations caused by possible obstructions. Specifically, a direct radio link between two nodes is said to exist if $L < L_{\text{th}}$, where L is the power loss and L_{th} represents the maximum loss tolerable by the communication system. In that case, the two nodes are said to be "*audible*". The threshold L_{th} depends on transmit power and receiver sensitivity. The power loss in decibel scale at distance d is expressed in the following form

$$L = k_0 + k_1 \ln d + s, \tag{2}$$

where k_0 and k_1 are constants, s is a Gaussian r.v. with zero mean, variance σ^2, which represents channel fluctuations. This channel model was also adopted in [21]. By considering an average transmission range as in [21], an average connectivity area of the sensor can be defined as

$$A_\sigma = \pi e^{\frac{2(L_{\text{th}} - k_0)}{k_1}} e^{\frac{2\sigma^2}{k_1^2}}. \tag{3}$$

4 Evaluation of the Area Throughput

The Area Throughput is mathematically derived through an intermediate step: we first consider the probability of successful data transmission by an arbitrary sensor node, when k nodes are present in the queried area. Then, the overall *Area Throughput* is evaluated based on this result.

4.1 Joint MAC/Connectivity Probability of Success

Let us consider an arbitrary sensor node that is located in the queried area A at a certain time instant. We aim at computing the probability that it can connect to one of the sinks deployed in A and successfully transmit its data sample to the infrastructure. Such an event is clearly related to connectivity issues (i.e., the sensor must employ an adequate transmitting power in order to reach the sink and not be isolated) and to MAC problems (i.e., the number of sensors which attempt at connecting to the same sink strongly affects the probability of successful transmission). For this reason, we define $P_{s|k}(x,y)$ as the probability of successful transmission conditioned on the overall number, k, of sensors present in the queried area, which also depends on the position (x,y) of the sensor relative to a reference system with origin centered in A. This dependence is due to the well-known border effects in connectivity [8].

In particular, we assume

$$\begin{aligned} P_{s|k}(x,y) &= E_n[P_{MAC}(n) \cdot P_{CON}(x,y)] \\ &= E_n[P_{MAC}(n)] \cdot P_{CON}(x,y), \end{aligned} \tag{4}$$

with E_x being the expectation operator with respect to variable x and where we separated the impact of connectivity and MAC on the transmission of samples. A packet will be successfully received by a sink if the sensor node is connected to at least one sink and if no MAC failures occur. We now analyze the two terms that appear in (4).

$P_{CON}(x,y)$ represents the probability that the sensor is not isolated (i.e., it receives a sufficiently strong signal from at least one sink), which is computed in [16] for a scenario analogous to the one considered here (e.g., squared and rectangular areas). This probability decreases as the sensor approaches the borders (border effects). Specifically, since the position of the sensor is in general unknown, $P_{s|k}(x,y)$ of (4) can be deconditioned as follows:

$$\begin{aligned} P_{s|k} &= E_{x,y}[P_{s|k}(x,y)] \\ &= E_{x,y}[P_{CON}(x,y)] \cdot E_n[P_{MAC}(n)]. \end{aligned} \tag{5}$$

It is also shown in [16] that border effects are negligible when $A_\sigma < 0.1A$. In this case the following holds:

$$P_{CON}(x,y) \simeq P_{CON} = 1 - e^{-\mu_{sink}}, \tag{6}$$

where $\mu_{sink} = \rho_0 A_\sigma = I A_\sigma / A$ is the mean number of audible sinks on an infinite plane from any position [21]. Throughout this paper we assume that connectivity is not affected by border effects. However, in case it is, the approach remains completely valid: only the computation of $E_{x,y}[P_{CON}(x,y)]$ requires greater efforts (see [16]).

$P_{MAC}(n)$, $n \geq 1$, is the probability of successful transmission when $n-1$ interfering sensors are present. It accounts for MAC issues and is treated in Section 5 for the particular case of the IEEE 802.15.4 standard, even though the

model is applicable to any MAC protocol. For now we only emphasize that it is a monotonic decreasing function of the number, n, of sensors which attempt to connect to the same serving sink. This number is in general a random variable in the range $[0, k]$. In fact, note that in (4) there is no explicit dependence on k, except for the fact that $n \leq k$ must hold. Moreover in our case we assume $1 \leq n \leq k$, as there is at least one sensor competing for access with probability P_{CON} (6).

In [22], Orriss *et al.* showed that the number of sensors uniformly distributed on an infinite plane that hear one particular sink as the one with the strongest signal power (i.e., the number of sensors competing for access to such sink) is Poisson distributed with mean

$$\bar{n} = \mu_s \frac{1 - e^{-\mu_{sink}}}{\mu_{sink}}, \tag{7}$$

with $\mu_s = \rho_s A_\sigma$ being the mean number of sensors that are audible by a given sink. Such a result is relevant toward our goal even though it was derived on the infinite plane. In fact, when border effects are negligible (i.e., $A_\sigma < 0.1A$) and k is large, n can still be considered Poisson distributed. The only two things that change are:

- n is upper bounded by k (i.e., the pdf is truncated)
- the density ρ_s is to be computed as the ratio k/A $[m^{-2}]$, thus yielding $\mu_s = k\frac{A_\sigma}{A}$.

Therefore, we assume $n \sim \text{Poisson}(\bar{n})$, with

$$\bar{n} = \bar{n}(k) = k\frac{A_\sigma}{A}\frac{1 - e^{-\mu_{sink}}}{\mu_{sink}} = k\frac{1 - e^{-IA_\sigma/A}}{I}. \tag{8}$$

Finally, by making the average in (5) explicit and by means of (6), we get

$$P_{s|k} = (1 - e^{-IA_\sigma/A}) \cdot \frac{1}{M} \sum_{n=1}^{k} P_{MAC}(n) \frac{\bar{n}^n e^{-\bar{n}}}{n!}, \tag{9}$$

where

$$M = \sum_{n=1}^{k} \frac{\bar{n}^n e^{-\bar{n}}}{n!} \tag{10}$$

is a normalizing factor.

4.2 Area Throughput

The amount of data samples generated by the network as response to a given query is equal to the number of sensors, k, that are present and active when the query is received. As a consequence, the average number of data samples-per-query generated by the network is the mean number of sensors, $\bar{k}$, in the queried area.

Now denote by G the *Offered Load*, that is the average number of data samples generated per unit of time, given by

$$G = \bar{k} \cdot f_q = \rho_s \cdot A \cdot \frac{1}{T_q} \text{ [samples/sec]}. \tag{11}$$

From (11) we have $\bar{k} = GT_q$.

The average amount of data received by the infrastructure per unit of time (*Area Throughput*), S, is given by:

$$S = \sum_{k=0}^{+\infty} S(k) \cdot g_k \text{ [samples/sec]}, \tag{12}$$

where $S(k) = \frac{k}{T_q} P_{s|k}$, g_k as in (1) and $P_{s|k}$ as in (9).

Finally, by means of (9), (10) and (11), equation (12) may be rewritten as

$$S = \frac{1 - e^{-IA_\sigma/A}}{T_q} \cdot \sum_{k=1}^{+\infty} \frac{\sum_{n=1}^{k} P_{MAC}(n) \frac{\bar{n}^n e^{-\bar{n}}}{n!}}{\sum_{n=1}^{k} \frac{\bar{n}^n e^{-\bar{n}}}{n!}} \cdot \frac{(GT_q)^k e^{-GT_q}}{(k-1)!}. \tag{13}$$

5 The IEEE 802.15.4 MAC Protocol

In this Section we introduce the two mathematical models used to derive the success probability, $P_{MAC}(n)$, when an IEEE 802.15.4 single-sink scenario, is considered. Both Beacon-Enabled and Non Beacon-Enabled modes are studied here (see the standard [5]).

In [17,18] an analytical model of the IEEE 802.15.4 Non Beacon-Enabled mode, was presented. For details on the protocol we refer to the standard as well. Here, we just want to underline that a maximum number of times a node can try to access the channel and perform the backoff algorithm, $NB_{max} = 4$, is imposed. According to this, there will be a maximum delay affecting a packet transmission. In [17,18] it is shown that, in case nodes transmit packets of size $D \cdot 10$ Bytes, the maximum interval of time between the instant at which a node receives the query and the instant at which the sink receives the end of the packet is equal to $(120 + D) \cdot T$. Moreover, since the time needed to transmit the query is equal to $6 \cdot T$ (being the query size 60 Bytes), the maximum delay with which the sink may receive a packet will be equal to $(126 + D) \cdot T$. Therefore, to ensure that all nodes have finished the CSMA/CA algorithm, the sink should set $T_q \geq (126 + D) \cdot T$. However, no constraints are imposed on the values of T_q in this modality, therefore lower values of T_q should be used (see Figure 2, above part). If a node receives a new query, when it is still trying to access the channel, the old packet will be lost, and the node will take a new sample and will start again the CSMA/CA algorithm.

For what concerns the Beacon-Enabled mode, the model described in [19] is used here to derive $P_{MAC}(n)$. According to this mode, the access to the channel is managed through a superframe, starting with a packet, called beacon, transmitted by the sink. In our scenario, the beacon includes the query. The superframe is composed of 16 slots and is divided into two parts: a Contention Access Period (CAP), during which nodes use a slotted CSMA/CA, and a Contention Free Period (CFP), containing a number of Guaranteed Time Slots (GTSs), that can be allocated by the sink to specific nodes (see Figure 2, below part). We denote as N_{GTS} the number of GTSs allocated by the sink to specific nodes in the PAN.

The superframe duration, SD, depends on the value of an integer parameter ranging from 0 to 14, called Superframe Order, SO, and given by:

$$SD = 16 \cdot 60 \cdot 2^{SO} T_s \,; \tag{14}$$

where $60 \cdot 2^{SO} T_s$ is the slot size, and T_s=16μsec is the symbol time. Since the queries coincide with beacons in this case, T_q must be equal to SD. Therefore, T_q may assume only a finite set of values, depending on SO. Note that by increasing SO, the time nodes are given to try to access the channel also increases, but the query frequency is smaller. The sink may allocate up to seven GTSs, and a GTS may occupy more than one slot. However, a sufficient portion of the superframe must remain for contention-based access. The minimum CAP duration is equal to 440 T_s. In case a node accessing the channel through the CSMA/CA algorithm does not succeed in accessing the channel by the end of the CAP portion, its packet will be lost.

By changing the packet size (i.e., D) and the slot size (i.e., SO), the number of slots occupied by each GTS and the maximum number of GTSs that can be allocated, vary. As an example, when $D = 2$ the packet occupies one slot, whatever SO and, therefore, the sink may allocate up to seven GTSs. In the case $D = 10$, instead, a packet occupies four slots when $SO = 0$, two slots when $SO = 1$, and one slot when $SO \geq 2$. Therefore, a maximum number of two, six and seven GTSs may be allocated in the three cases, respectively.

In this Section we show some results obtained through our mathematical model, considering a single-sink scenario, where n sensors transmit data to the sink through a direct link, with no connectivity problems. A finite state transition diagram is used to model sensor nodes states, in both cases Beacon- and Non Beacon-Enabled mode. Through the analysis of this diagram the probability that a given sensor successfully transmits its packet, $P_{MAC}(n)$, is evaluated. We do not report here the expression of this probability, owing to its complexity, but we refer to [18, 17] for the Non Beacon-Enabled case and to [19] for the Beacon-Enabled case, where details on formulae are given and where a validation of the model, through comparison with simulations, is provided for $n \leq 50$. The probability $P_{MAC}(n)$ obtained in these works, can be used in (13) for the evaluation of S.

In Figures 3 and 4 P_{MAC} as functions of n, for different values of SO, having fixed $D = 2$ and $D = 10$, are shown, respectively. The cases of no GTSs allocated and N_{GTS} equal to the maximum number of GTSs allocable, are considered. As

explained above, this maximum number depends on the values of D and SO. In these Figures the Beacon-Enabled mode is analysed. As we can see, P_{MAC} decreases monotonically (for $n > 1$ when $N_{GTS} = 0$ and for $n > N_{GTS}$ when $N_{GTS} > 0$), by increasing n, since the number of sensors competing for the channel increases. In both Figures, once we fix SO, by increasing N_{GTS}, P_{MAC} increases, since less nodes have to compete for the channel. Moreover, once we fix N_{GTS}, by increasing SO, P_{MAC} gets larger, since increases the CAP size and nodes have more time to try to access the channel.

In Figure 5 $P_{MAC}(n)$ as functions of n, for different values of D and T_q, is shown, considering a Non Beacon-Enabled network. As we can see, a decrease of T_q, results in a decreasing of P_{MAC}, since nodes have less time to access the channel. However, the decreasing of T_q brings to an increasing of the queries frequency, and this impacts on the *Area Throughput* (see Section 6).

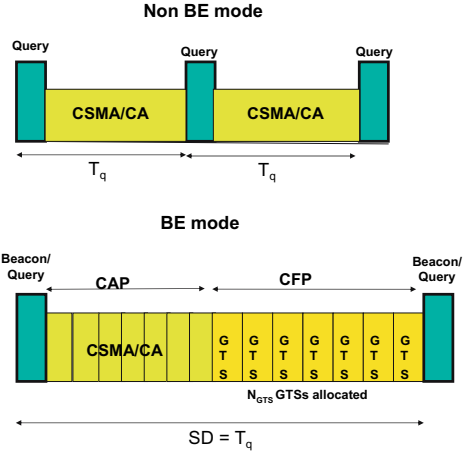


Fig. 2. Above part: The IEEE 802.15.4 Non Beacon-Enabled mode. Below part: The IEEE 802.15.4 Beacon-Enabled mode.

6 Numerical Results

In this Section the behavior of the *Area Throughput* as a function of the *Offered Load*, G, for the Beacon- and the Non Beacon-Enabled modes, considering different values of D, SO, N_{GTS}, T_q and different connectivity levels, is shown. Validation of the framework via simulation is not reported here for a matter of space. However, the validation of connectivity and MAC models separately may be found in [23] and [18,19], respectively.

Let us consider a square domain, having area $A = 10^6$ m^2, where an average number of 10 sinks are distributed according to a PPP ($I = 10$). We also set $k_0 = 40$ dB, $k_1 = 13.03$, $\sigma = 4$ dB (the values are taken from experimental measurements made on the field with Freescale devices [24]) and $L_{th} = 107$ dB.

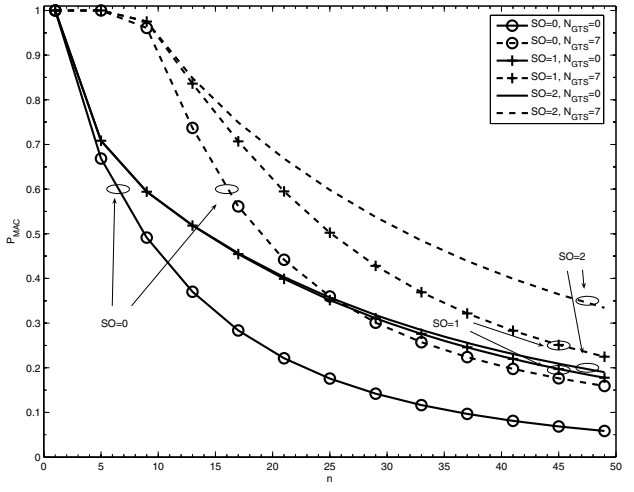


Fig. 3. $P_{MAC}(n)$ as a function of n, in the Beacon-Enabled case, for different values of SO and N_{GTS}, having fixed $D = 2$

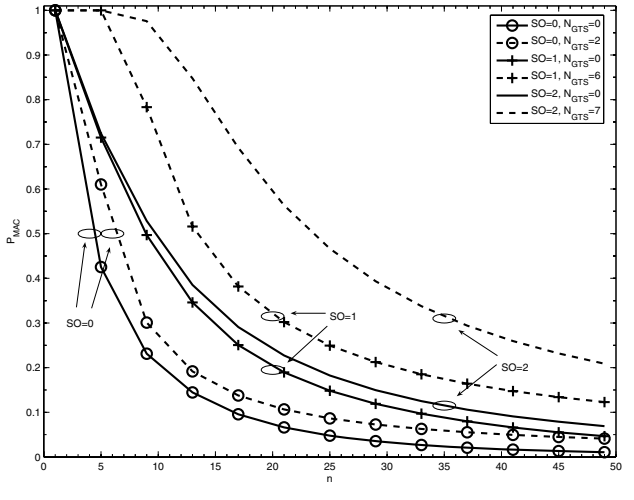


Fig. 4. $P_{MAC}(n)$ as a function of n, in the Beacon-Enabled case, for different values of SO and N_{GTS}, having fixed $D = 10$

In Figures 6 and 7, S as a function of G, when varying SO, N_{GTS} and T_q, for $D = 2$ and $D = 10$, is shown, respectively. The input parameters that we entered give a connection probability $P_{CON} = 0.89$. Both Beacon- and Non Beacon-Enabled modes are considered. In both Figures we note that, once SO is fixed (Beacon-Enabled case), an increase of N_{GTS} results in an increment of S, since P_{MAC} increases. Moreover, once N_{GTS} is fixed, there exists a value of SO

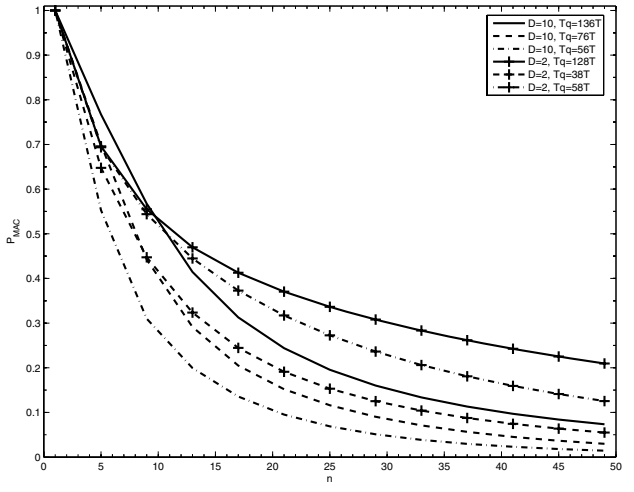


Fig. 5. $P_{MAC}(n)$ as a function of n, in the Non Beacon-Enabled case, for different values of T_q and D

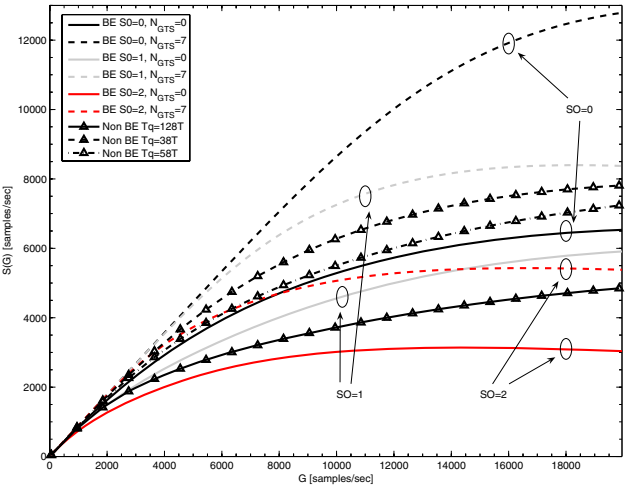


Fig. 6. S as a function of G, for the Beacon- and Non Beacon-Enabled cases, by varying SO, N_{GTS} and T_q, having fixed $D = 2$

maximising S. When $D = 2$, an increase of SO results in a decrement of S since, even though P_{MAC} gets greater, the query interval is longer and, therefore, the number of samples per second received by the sink decreases. On the other hand, when $D = 10$ and all possible GTSs are allocated, the optimum value of SO is 1. This is due to the fact that, having large packets, when $SO = 0$ too many packets are lost, owing to the short duration of the superframe. However, when

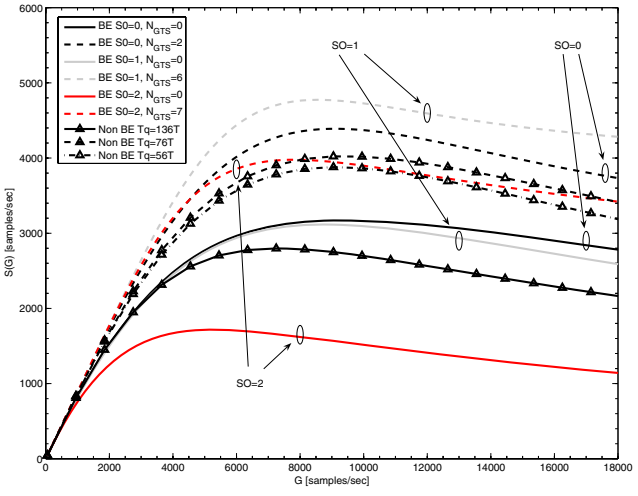


Fig. 7. S as a function of G, for the Beacon- and Non Beacon-Enabled cases, by varying SO, N_{GTS} and T_q, having fixed $D = 10$

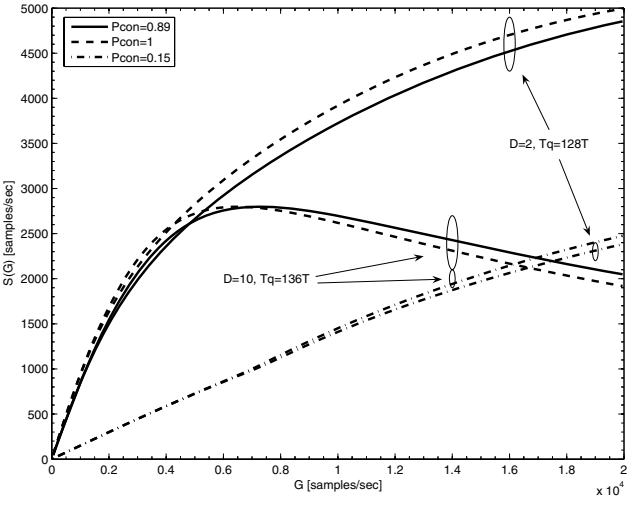


Fig. 8. S as a function of G, in the Non Beacon-Enabled case, for different values of D and P_{CON}, having fixed T_q to the maximum delay

$N_{GTS} = 0$ the best case is, once again, $SO = 0$, since in this case MAC losses are approximately the same obtained in the case of $SO = 1$ (see Figure 4), which, however, brings to a higher query interval. In conclusion, we can deduce that the use of GTSs is always advantageous, and that there exists an optimum value of SO maximising S, which depends on D and N_{GTS}.

Concerning the Non Beacon-Enabled case, in both Figures we note that, by decreasing T_q, S gets larger even though P_{MAC} decreases, since, once again, the MAC losses are balanced by larger values of f_q.

By comparison of Figures 6 and 7, we note that, once the *Offered Load*, G, is fixed, S gets notably smaller when D increases. S, in fact, is expressed in terms of samples/sec received by the sink, and not in Bytes/sec. Therefore, once T_q is fixed, by increasing D, P_{MAC} gets smaller. On the other hand, by increasing D, the maximum value of S is reached for lower values of G. This means that, when D is small, the maximum value of S is reached at the cost of deploying more sensors.

Finally, we show the effects of connectivity on the *Area Throughput*. When P_{CON} is less than 1, only a fraction of the deployed nodes has a sink in its vicinity. In particular, an average number, $\bar{k} = P_{CON} G T_q / I$, of sensors compete for access at each sink. In Figure 8 we consider the Non Beacon-enabled case with $D = 2$, $T_q = 128T$ and $D = 10$, $T_q = 136T$. When $D = 10$, $T_q = 136T$, for high *Offered Loads* the area throughput tends to decay, since packet collisions dominate. Hence, by moving from $P_{CON} = 1$ to $P_{CON} = 0.89$, we observe a slight improvement due to the fact that a smaller average number of sensors tries to connect to the same sink. Conversely, when $D = 2$, $T_q = 128T$, S is still increasing with G, then by moving from $P_{CON} = 1$ to $P_{CON} = 0.89$, we just reduce the useful traffic. Furthermore, when $P_{CON} = 0.15$, the *Offered Load* is very light, so that we are working in the region where $P_{MAC}(D = 2, T_q = 128T) < P_{MAC}(D = 10, T_q = 136T)$ (see Fig. 5), resulting in a slightly better performance of the case with $D = 2$. Thus we conclude that the effect of lowering P_{CON} results in a stretch of the curves reported in the previous plots.

7 Conclusions

An IEEE 802.15.4 standard compliant multi-sink WSN where sensor nodes transmit their packets to a sink selected among many, is studied. A mathematical framework is developed to evaluate the *Area Throughput*, that is the amount of samples per second successfully transmitted to the sinks. The behavior of the *Area Throughput* for Beacon-Enabled and Non Beacon-Enabled networks, by considering different packet sizes, number of GTSs allocated, and queries frequency, is shown. Results show that the use of GTSs improves performance of the Beacon-Enabled mode and that thanks to these GTSs this mode outperforms the Non Beacon-Enabled mode. Moreover, results show that there exists a value of SO maximising the *Area Throughput* and this value depends on D and N_{GTS}. Finally, the effects of the connectivity on the *Area Throughput* are evaluated. Results show that when connectivity decreases, the number of sensors that must be deployed to obtain large *Area Throughput*, increases.

Acknowledgment

This work was supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715).

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. IEEE Communications Magazine, 102–114 (2002)
2. Verdone, R., Dardari, D., Mazzini, G., Conti, A.: Wireless sensor and actuator networks, 1st edn. Elsevier, Amsterdam (2008)
3. Dardari, D., Conti, A., Buratti, C., Verdone, R.: Mathematical evaluation of environmental monitoring estimation error through energy-efficient Wireless Sensor Networks. IEEE Transaction on Mobile Computing 6(7), 790–803 (2007)
4. Buratti, C., Verdone, R.: A Hybrid Hierarchical Architecture: From a Wireless Sensor Network to the Fixed Infrastructure. In: IEEE EW 2008, Prague, Czech Republic, pp. 22–25 (June 2008)
5. IEEE 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE, Wei Ye (2003)
6. Stuedi, P., Chinellato, O., Alonso, G.: Connectivity in the presence of Shadowing in 802.11 Ad Hoc Networks. In: IEEE WCNC 2005 (2005)
7. Buratti, C., Verdone, R.: On the of Cluster Heads minimising the Error Rate for a Wireless Sensor Network using a Hierarchical Topology over IEEE 802.15.4. IEEE PIMRC 2006, Helsinki, FI, September 11-14 (2006)
8. Bettstetter, C.: On the minimum node degree and connectivity of a wireless multihop network. In: Proc. ACM Symp. on Mobile Ad Hoc Networks and Comp. (Mobihoc) (June 2002)
9. Vincze, Z., Vida, R., Vidacs, A.: Deploying multiple sinks in multi-hop wireless sensor networks. In: IEEE International Conference on Pervasive Secrices, July 15-20, 2007, pp. 55–63 (2007)
10. Kim, T.O., Park, J.S., Kim, K.J., Choi, B.D.: Analytical Model of IEEE 802.15.4 Non-beacon Mode with Download Traffic by the Piggyback Methods. In: International Federation for Information Processing 2007, pp. 435–444 (2007)
11. Misic, J., Shafi, S., Misic, V.B.: Maintaining Reliability Through Activity Management in an 802.15.4 Sensor Cluster. IEEE Transactions on Vehicular Technology 55(3), 779–788 (2006)
12. Pollin, S., Ergen, M., Ergen, S.C., Bougard, B., Van der Pierre, L., Catthoor, F., Moerman, I., Bahai, A., Varaiya, P.: Performance Analysis of Slotted Carrier Sense IEEE 802.15.4 Medium Access Layer. In: Proceeding of GLOBECOM 2006, San Francisco, California, November 27 - December 1 (2006)
13. Park, T.R., Kim, T.H., Choi, J.Y., Choi, S., Kwon, W.H.: Throughput and energy consumption analysis of IEEE 802.15.4 slotted CSMA/CA. IEEE Electronics Letters 41(18), 1017–1019 (2005)
14. Chen, Z., Lin, C., Wen, H., Yin, H.: An analytical model for evaluating IEEE 802.15.4 CSMA/CA protocol in Low rate wireless application. In: Proceedings of IEEE AINAW 2007 (2007)
15. Bianchi, G.: Performance Analysis of the IEEE 802.11 Distributed Coordination Function. IEEE Journal on Selected Areas in Communications 18(3) (March 2000)
16. Fabbri, F., Verdone, R.: A statistical model for the connectivity of nodes in a multisink wireless sensor network over a bounded region. In: IEEE EW 2008, Prague, Czech Republic, pp. 22–25 (June 2008)
17. Buratti, C., Verdone, R.: Performance Analysis of IEEE 802.15.4 Non-Beacon Enabled Mode. IEEE Transactions on Vehicular Technologies (TVT) (to appear)

18. Buratti, C., Verdone, R.: A Mathematical Model for Performance Analysis of IEEE 802.15.4 Non-Beacon Enabled Mode. In: IEEE EW 2008, Prague, Czech Republic, June 2008, pp. 22–25 (2008)
19. Buratti, C.: A Mathematical Model for Performance of IEEE 802.15.4 Beacon-Enabled Mode. In: IEEE ICC 2009 (submitted, 2009)
20. Verdone, R., Fabbri, F., Buratti, C.: Area Throughput for CSMA Based Wireless Sensor Network. In: IEEE PIMRC 2008, September 15-18, Cannes, France (2008)
21. Orriss, J., Barton, S.K.: Probability distributions for the number of radio transceivers which can communicate with one another. IEEE Trans. Commun. 51(4), 676–681 (2003)
22. Verdone, R., Orriss, J., Zanella, A., Barton, S.: Evaluation of the blocking probability in a cellular environment with hard capacity: a statistical approach. In: Personal, Indoor and Mobile Radio Communications (PIMRC 2002), September 15-18, 2002, vol. 2 (2002)
23. Fabbri, F., Buratti, C., Verdone, R.: A Multi-Sink Multi-Hop Wireless Sensor Network Over a Square Region: Connectivity and Energy Consumption Issues. In: Wireless Mesh and Sensor Networks Workshop at GLOBECOM 2008, New Orleans, LA, USA, 30 November-4 December (2008)
24. Freescale, Freescale semiconductor's mc13192 developer's kit, http://www.freescale.com/webapp/sps/site/prodsummary.jspcode=13193EVB

Experimental Study of the Impact of WLAN Interference on IEEE 802.15.4 Body Area Networks*

Jan-Hinrich Hauer, Vlado Handziski, and Adam Wolisz

Telecommunication Networks Group
Technische Universität Berlin, Germany
{hauer,handzisk,wolisz}@tkn.tu-berlin.de

Abstract. As the number of wireless devices sharing the unlicensed 2.4 GHz ISM band increases, interference is becoming a problem of paramount importance. We experimentally investigate the effects of controlled 802.11b interference as well as realistic urban RF interference on packet delivery performance in IEEE 802.15.4 body area networks. Our multi-channel measurements, conducted with Tmote Sky sensor nodes, show that in the low-power regime external interference is typically the major cause for substantial packet loss. We report on the empirical correlation between 802.15.4 packet delivery performance and urban WLAN activity and explore 802.15.4 cross-channel quality correlation. Lastly, we examine trends in the noise floor as a potential trigger for channel hopping to detect and mitigate the effects of interference.

Keywords: Body Area Networks, IEEE 802.15.4, Interference.

1 Introduction

Body Area Networks (BANs) allow monitoring of the human body with detail and pervasiveness that is opening new application opportunities in domains ranging from personalized health-care and assisted living to sport and fitness monitoring [1]. In these domains the wireless telemetry was traditionally based either on proprietary communication technologies or on standardized solutions with significant licencing overhead and limited geographic availability. With the introduction of the IEEE 802.15.4 standard [8] and its focus on low data rates, low power consumption, reduced complexity and device size, an alternative emerged that matches the specific requirement of a BAN platform quite well.

Although 802.15.4 technology has rapidly matured and become the basis of several commercial products, there is still a level of uncertainty whether it can meet the stringent QoS requirements typical for some BAN applications under more challenging operating conditions. These concerns especially pertain to the coexistence with other major users of the unlicensed 2.4 GHz ISM band, notably

* This work has been partially supported by the European Commission under the contracts FP7-2007-IST-2-224053 (CONET) and FP6-2005-IST-5-033506 (ANGEL).

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 17–32, 2009.
© Springer-Verlag Berlin Heidelberg 2009

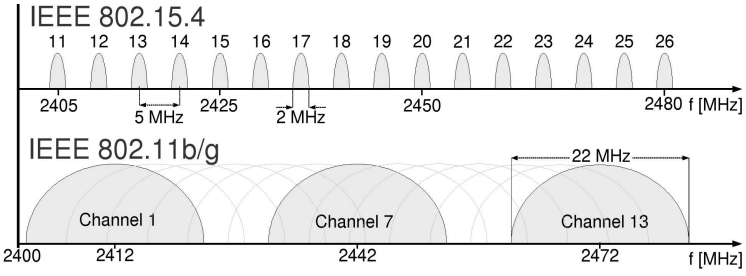


Fig. 1. IEEE 802.15.4 and 802.11 spectrum usage in the 2.4 GHz ISM band. The availability of channels is regulated per country.

IEEE 802.11 (WLAN) [10] and IEEE 802.15.1 (Bluetooth) [9]. Due to their virtual omnipresence and comparably high transmit power (20 dBm in Europe) WLANs pose a particular challenge.

Fig. 1 shows the spectrum usage of the two technologies in the 2.4 GHz ISM band. Despite interference mitigation mechanisms like DSSS and "listen-before-send" incorporated in both standards, it is well established that their mutual interference can result in notable deterioration of packet delivery performance. Although previous studies have treated this problem both from analytical and experimental side (Sect. 6), none of them has taken into consideration the specific characteristics and operational features of the BAN domain in terms of topology configuration, mobility and radio duty cycle under *realistic interference scenarios* in typical urban environments.

Our work targets this unexplored area. We present a measurement setup that allows capturing of a large subset of the parameter space with detail that was previously not reported. It supports mobile long-term monitoring of interference effects using symmetric communication and variable transmit power on all sixteen IEEE 802.15.4 channels in the 2.4 GHz band. We (1) report on multi-channel measurements from a controlled environment as well as from different urban environments, (2) demonstrate empirical correlation between 802.15.4 packet delivery performance and "real-life" WLAN activity and (3) explore 802.15.4 cross-channel quality correlation and trends that may be used as a potential trigger for channel hopping.

We believe that our study is an important step towards a realistic assessment of how WLAN interference can affect IEEE 802.15.4 BANs and towards the development of schemes for interference detection and mitigation.

The rest of the paper is structured as follows: in Sect. 2 we describe our experimental setup and provide a definition of the relevant metrics. We present the results of a set of baseline experiments in Sect. 3 and report on a representative sample of our dataset from an urban environment in Sect. 4. In a "first cut" evaluation we analyze in Sect. 5 the empirical traces for cross-channel quality correlation and trends in the noise floor. In Sect. 6 we discuss related work and present our conclusions and plans for future work in Sect. 7.

2 Experimental Setup

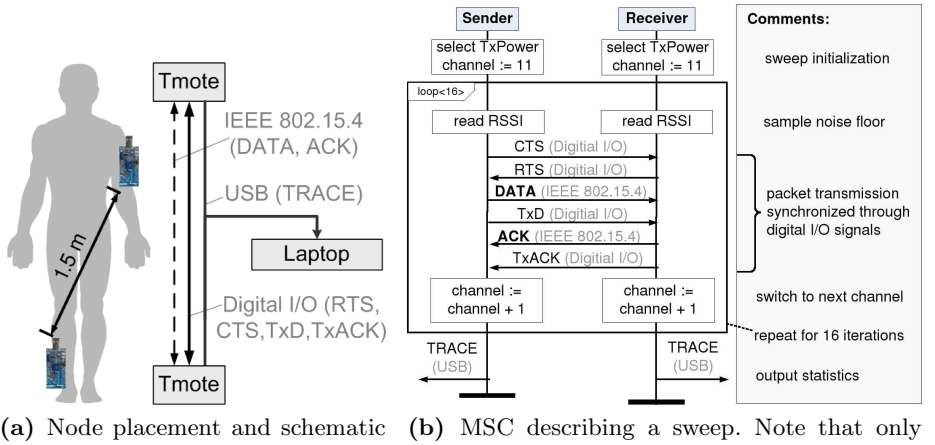
This section introduces our measurement platform and provides a definition of the relevant metrics.

2.1 Measurement Platform

Our measurements are performed with Tmote Sky [13] (Telos Rev. B) sensor nodes, which are equipped with the IEEE 802.15.4-compliant Texas Instruments CC2420 transceiver [2]. The CC2420 operates in the 2.4 GHz ISM band, uses O-QPSK modulation and has a data rate of 250 kbps. A packet can be transmitted on one of 16 channels which are spaced 5 MHz apart and occupy frequencies 2405 MHz - 2480 MHz as shown in Fig. 1.

Our setup consists of two nodes, each node is placed in a thin plastic enclosure and strapped to a person: one to the left upper arm, the other on the right shin just above the ankle, resulting in a relative distance of about 1.5 m (Fig. 2a, left). When the person stands still both nodes have the same alignment and both surface areas are facing in the same horizontal direction. However, in all experiments, the test person is walking at an even speed of about 1.2 m/s (common walking speed). Our setup introduces two auxiliary wired channels: one to synchronize the transmissions on the IEEE 802.15.4 channel; and a second for streaming measurement results to a laptop. This is schematically shown in Fig. 2a (right) and explained in the following.

Packet Transmission. Our measurement software accesses the CC2420 radio directly, there is no MAC layer involved and all packets are sent immediately (without clear channel assessment, CCA). Both nodes continuously iterate over the 16 channels, exchanging one `DATA` and acknowledgement (`ACK`) packet per



(a) Node placement and schematic measurement setup. The laptop is carried in a backpack.

(b) MSC describing a sweep. Note that only DATA and ACK packets are exchanged over the wireless channel.

Fig. 2. Measurement setup and Message Sequence Chart (MSC) for a single sweep

channel. We call an iteration over all 16 channels – involving 32 packets – a *sweep*. During a sweep the roles of the nodes are fixed: one node sends the `DATA` packets, the other node sends the `ACK` packets; after every sweep the roles are swapped. We use acknowledgements, because this is common to many IEEE 802.15.4 networks, and we let the nodes swap roles, because this allows us to (better) evaluate link (a)symmetry.

The CC2420 supports different transmission power levels, the datasheet documents 8 discrete levels ranging from −25 dBm to 0 dBm [2]. The relevant `TXCTRL` register of the radio, however, accepts 32 different values. The radio manufacturer confirmed to us that all 32 values are valid, however, "the relation between the register setting and the output power is not linear". We experimentally determined an output power of roughly −42 dBm when the `TXCTRL.PA_LEVEL` is set to the value of 2.[1] In our study we then used three different output power levels of −10 dBm, −25 dBm and −42 dBm, alternating every sweep. The MSC in Fig. 2b shows the sequence of operations performed during a sweep.

Synchronization. Whenever a packet is transmitted by one node, the other node's radio is in receive mode, both nodes have tuned to the same channel, and they use the same level of transmission power for the `DATA` and `ACK` packet. This requires very careful synchronization. One option is to perform synchronization through the arrival times of `DATA` and `ACK` packets, but since our experiments are conducted in an environment of possibly strong RF interference we rejected such "in-band" synchronization, because it would require unknown (conservative) guard times to counter the clock drift in case of successive packet losses. Instead we perform synchronization over digital I/O signals through pins exposed on the Tmote Sky expansion header: with a shielded cable we interconnect the two microcontrollers via four digital I/O ports. To increase robustness we use differential signalling (complementary signals sent on two separate wires) and thus allow four different signals, which is sufficient to synchronize the transmission of `DATA` and `ACK` packets. The sequence of exchanged digital I/O signals is shown in Fig. 2b, the vocabulary of messages and signals is listed in Table 1.

Data Logging. During our study the sensor nodes collect large amounts of statistics that, due to limited memory, cannot be stored on the nodes themselves. Instead the nodes are connected and continuously output the results to a laptop through the USB interface of the Tmote Sky, which also serves as their power supply. The laptop is carried in a backpack by the same person that wears the two sensor nodes and is also used to monitor 802.11b/g traffic on selected WLAN channels during the experiments.

[1] We chose 38 sender/receiver combinations from a batch of 10 Tmote Sky nodes and measured RSSI for different documented as well as the undocumented register settings in a low-interference environment based on a fixed node placement. Assuming a linear relationship between output power and RSSI, we used the documented values in combination with the measured average RSSI as anchors for a linear regression to determine the unknown transmission power level based on the corresponding measured RSSI value.

Table 1. Messages and signals exchanged during the measurements

Message/ Signal	Channel	Description
DATA	802.15.4	DATA packet (MPDU of 36 byte)
ACK	802.15.4	ACK packet (MPDU of 5 byte)
RTS	Digital I/O	Sender requests to send a DATA packet
CTS	Digital I/O	Receiver is ready to receive DATA packet
TxD	Digital I/O	Sender has sent a DATA packet
TxACK	Digital I/O	Receiver has sent an ACK packet
TRACE	USB	Measurement results for a sweep over 16 channels

Sweep-time Performance. In our setup the time required for a sweep over 16 channels is about 87 ms, which results in around 12 sweeps (384 packets) per second including streaming the measurement results over USB. This is achievable because we increase the Tmote Sky CPU frequency to the maximum of 8 MHz and because synchronization over digital I/O is very fast. We took particular care to minimize the impact of streaming the statistics over the USB on the actual measurement and its periodic workflow: most operations related to the transmission of 802.15.4 packets and synchronization occur in interrupt context, while sending serial packets over USB is divided in many small blocks of code that are executed in non-interrupt context. All outgoing/incoming packets are timestamped. After the measurement we use the hardware generated timestamps of successful transmissions as anchors, perform a linear regression to cancel out the clock drift, and obtain precise timing information to verify that in our setup 802.15.4 data packets are indeed transmitted periodically with an average interarrival time of around 5.5 ms. For example, in the experiment described in Sect. 4.1 we determine an average interarrival time of 5.43 ms (maximum: 6.45 ms, minimum: 4.39 ms).

2.2 Additional Hardware

We use a portable *Wi-Spy 2.4x* USB spectrum analyzer to verify that the baseline measurements are conducted in an environment of negligible external RF interference. The two laptops that generate controlled 802.11b traffic (Sect. 3.2) use Intel PRO/Wireless 2100 network interface cards. In some of our measurements we also monitor 802.11b/g traffic using a PC card based on an Atheros chipset plugged into the laptop that collects the measurement results.

2.3 Metrics

In our study all 802.15.4 `DATA` packets are acknowledged and a transmission is defined as *successful* if both, `DATA` and `ACK` packet, were received without errors. Correspondingly, a transmission has *failed* if either `DATA` or `ACK` packet (or both) were corrupted (CRC check failed). Whenever we report on moving averages,

the average is calculated over either 10 or 100 transmissions for a particular channel and transmission power level. Because a sweep takes about 87 ms and we use three different transmission power levels alternating every sweep, this corresponds to a time window of about 2.6 s (10 transmissions) or 26 s (100 transmissions).

The CC2420 radio adds to every received packet the Receive Signal Strength Indicator (RSSI) level and a Link Quality Indication (LQI) value. We use the formula in the CC2420 datasheet [2] to convert the exported RSSI value to dBm. The LQI value from the CC2420 "represents a measurement of correlation between the received [and the determined] chip"[2] and we always report the raw LQI values ranging from about 110 (maximum quality) to 50 (worst quality). In addition to per-packet RSSI and LQI we measure the noise floor in between transmissions by reading the CC2420 RSSI register, which we hereafter call SSI_{noise}.

3 Baseline Measurements

We begin our study with a set of baseline measurement conducted outdoors in a large park, an environment of negligible external RF interference, as verified with the help of a portable spectrum analyzer. The results are intended to give some confidence in the performance of our setup, to reveal that — at least in this environment — the effects of mobility are virtually negligible and to show possible effects of 802.11b interference on 802.15.4 link quality.

3.1 Low Interference

In our first measurement the test person takes a 35 minute walk while the BAN measures packet loss, noise floor, RSSI and LQI over the 16 different channels. Out of the total 390.096 transmission only 2 failed (one at −42 dBm, the other at −25 dBm). This is negligible and indicates that with our setup a transmission power of −42 dBm is in principle sufficient in this kind of environment. We observe only small variance in RSSI, the maximum standard deviation for RSSI on any channel for any out of three given transmission power was 1.54 dBm. However, at −42 dBm transmission power the RSSI is usually around −88 dBm, which is close to the −94 dBm sensitivity threshold specified in the CC2420 datasheet. LQI varies a little more, in particular at −42 dBm transmission power, where we observe a maximum standard deviation of 3.10 for channel 11 on one node. The noise floor was very stable at −99 dBm on both nodes.

3.2 Controlled 802.11b Interference

We are interested in how a nearby 802.11 network can affect the link quality of the 16 different 802.15.4 channels. In this experiment we set up two laptops to form an 802.11b ad-hoc network and start a large file transfer from one to the

[2] One 802.15.4 symbol is mapped to a sequence of 32 chips resulting in a nominal chip rate of 2.0 Mchip/s.

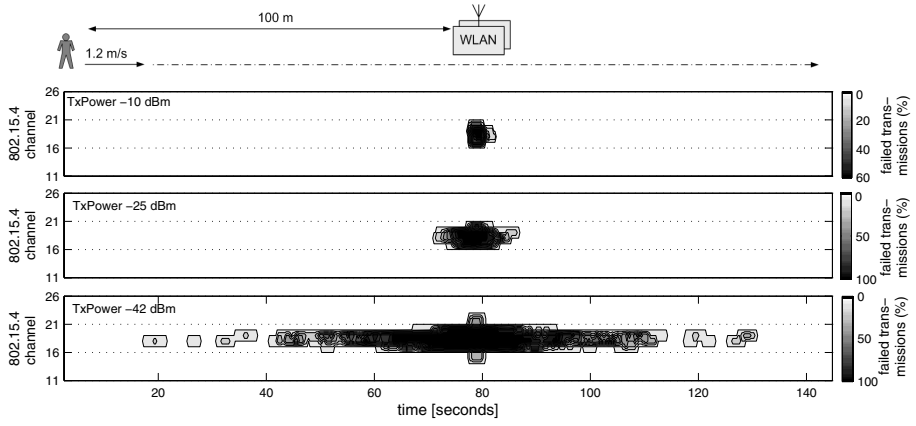


Fig. 3. Failed 802.15.4 transmissions while walking past two 802.11b stations transmitting at maximum rate on 802.11 channel 7. For every 802.15.4 channel the transmissions are averaged over a window of 10 transmissions, and 802.15.4 packets are transmitted with either −10 dBm (top), −25 dBm (middle) or −42 dBm (bottom) transmission power, alternating every sweep. The total distance covered is 180 m, the 802.11 network is located at 100 m distance from the starting point.

other. Both laptops are placed close to each other on the ground, and generate heavy traffic on 802.11 channel 7 at 11 Mbit/s. The experiment is simple: our test person first stands 100 m away from the 802.11 network, at time $t = 0$ s starts walking on a straight line towards it, passes 1 m by the two laptops (at about $t = 80$ s) and continues walking on the same straight line. The BAN measures the number of failed 802.15.4 transmissions, the changes in noise floor and per-packet RSSI and LQI. After the experiment we sort the measurement results by the transmission power level, and produce a contour plot, respectively, showing failed transmission averaged over a window of 10 transmissions (2.6 s) for each channel. The result can be seen in Fig. 3.

The 802.11b network temporarily caused significant packet loss: at −10 dBm transmission power transmissions failed only at close distance (a few meters), at −25 dBm losses occurred within about ± 10 m, and at −42 dBm the first transmissions failed at more than 75 m distance. However, packets were lost only on channels that are close to 2442 MHz, the center frequency of 802.11 channel 7, that is mainly on 802.15.4 channels 17 to 20 (compare Fig. 1). According to the 802.11 standard, at ± 11 MHz from the center frequency, the radiated energy must be 30 dB lower than the maximum signal level; still, when the 802.15.4 network transmitted at −42 dBm even channels 15, 16, 21 and 22 suffered short-term losses at close distance.

The results can be interpreted in line with the SINR model: from the perspective of the 802.15.4 BAN the 802.11b laptops generate interference, which decays (non-linearly) with distance. When approaching the interferers, at a certain distance the ratio of received power in the 802.15.4 signal to the power of

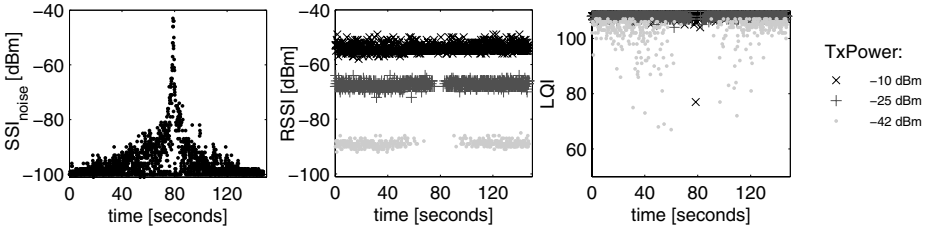


Fig. 4. Noise floor (left), RSSI (middle) and LQI (right) on channel 18 over time, extracted from the traces belonging to the experiment shown in Fig. 3

the interference is too low for the CC2420 radio to correctly decode the symbols and packets are lost.

Fig. 4 shows the dynamics in the noise floor, RSSI and LQI. The figure only shows the results for 802.15.4 channel 18, because it was one of the most affected by packet loss. Naturally RSSI and LQI are only available for received packets, but the respective graphs (middle and right in Fig. 4) give some first insight in temporal trends around the losses.

As expected, the noise floor increases with smaller distance to the 802.11b network; however, even at very close distance (around $t = 80\,\mathrm{s}$), we see a range of different SSI_{noise} values, some as low as $-99\,\mathrm{dBm}$. A possible explanation is that the 802.11 stations are not permanently transmitting. For example, there are at least short Inter-Frame Spaces (IFS) between 802.11 packets during which the channel is idle. The CC2420 averages a single SSI_{noise} reading over $128\,\mu\mathrm{s}$, and it can thus happen that a sample is taken while the channel is (partially) idle. This indicates that a single SSI_{noise} value is rather unreliable for determining presence of an interferer.

RSSI seems almost unaffected by the 802.11 traffic, but LQI shows higher variance as distance to the 802.11 network decreases, especially at $-42\,\mathrm{dBm}$ transmission power. This is understandable since LQI represents a measure of correlation between 802.15.4 chips: single chips may be corrupted by 802.11 interference while the symbols are still correctly decoded.

4 Urban Measurement Campaign

We made measurements in three different environments in the city of Berlin, Germany: at a shopping street, in a central residential area and in an office area. During all measurements the test person was walking outdoors on the urban streets using the same setup as described previously. A single measurement typically lasted around 30 minutes. As a case study in this section we report on one such measurement in detail, in the next Sect. 5 we present an evaluation of the empirical traces from all measurements.

The measurement described in the rest of this section was made at a central urban shopping street. Buildings were located on either side of the roughly 30 m wide street, which was moderately frequented by cars and other pedestrians on

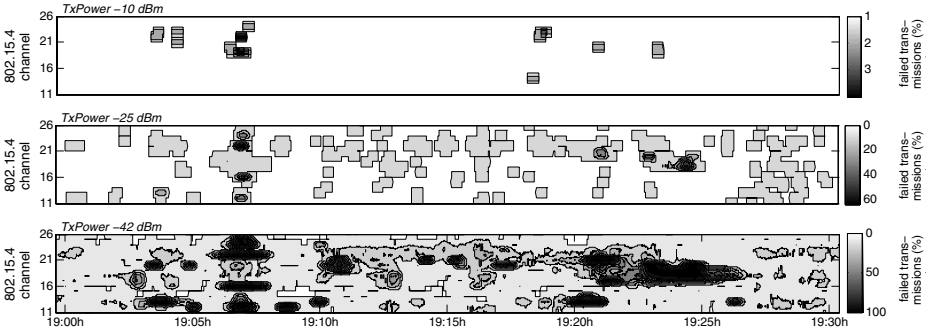


Fig. 5. Failed 802.15.4 transmissions while walking along an urban shopping street. For every 802.15.4 channel the transmissions are averaged over a window of 100 transmissions (26 s), and packets are sent with three different transmission power levels: −10 dBm (top), −25 dBm (middle) or −42 dBm (bottom), alternating every sweep.

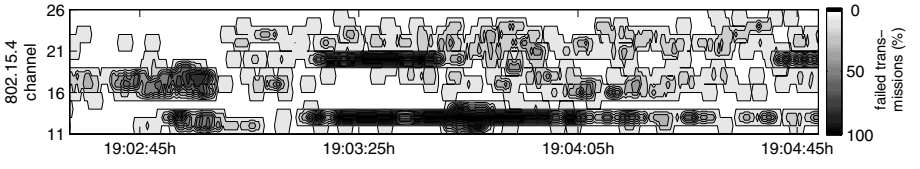


Fig. 6. Two minute excerpt from bottom Fig. 5, averaged over 10 transmissions

a weekday evening at 7 p.m. The test person took a 30 minute walk outdoors along the pavement passing by shops, coffee bars and offices as well as other pedestrians. The walk was one-way and close to straight-line at even walking speed (stopping only at red traffic lights).

4.1 Transmission Failures

Fig. 5 shows failed 802.15.4 transmissions averaged over 100 transmissions per channel (about 26 s) sorted by transmission power.

The losses for −10 dBm transmission power (top) are negligible, even at −25 dBm (middle) we never see more than 60 % loss within a window of 26 s on any channel. At −42 dBm (bottom) transmissions failed more frequently and some channels were temporarily completely blocked. The figure suggests that losses were not completely random. Instead they showed some correlation in time and frequency, sometimes lasting for a few tens of seconds up to multiple minutes and spanning over multiple consecutive 802.15.4 channels.

Fig. 6 shows a 2 minute excerpt of the bottom Fig. 5 at around 19:03 h. In this figure transmissions are averaged over a window of 10 transmissions (2.6 s) as in the baseline measurement shown in Fig. 3. When comparing these two figures we find that in Fig. 6 the pattern at around 19:03 h to 19:04 h on channels 13 closely resembles the 802.11b "footprint" in Fig. 3, which suggests that these losses might have been caused by 802.11 traffic.

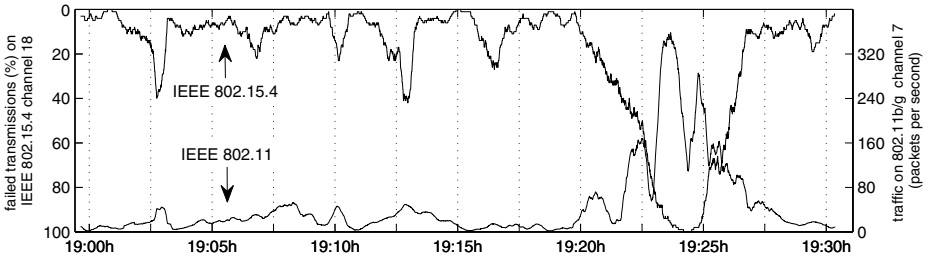


Fig. 7. 802.15.4 transmission failures on channel 18 using transmission power −42 dBm (left Y-Axis) and 802.11b/g traffic on WLAN channel 7 (right Y-Axis) averaged over a window of about 26 s

4.2 Correlation with 802.11b/g Traffic

During the experiments the 802.11b/g card of the laptop that collected the statistics from the nodes was set to passive monitoring mode so that all 802.11 traffic on a given channel was captured. We tuned the card to 802.11 channel 7, because it is one of the most commonly used. In this way we measured two things in parallel: 802.15.4 failures on all channels and 802.11b/g traffic on 802.11 channel 7.

In Fig. 7 one can see both, failures on 802.15.4 channel 18 using transmission power −42 dBm and the number of received 802.11 packets on channel 7, averaged over 100 transmissions in the 802.15.4 network (26 s). As shown in Fig. 1, 802.11 channel 7 completely overlaps with 802.15.4 channel 18. The results appear (negatively) correlated both, visually and statistically. The empirical correlation coefficient is r = -0.89, which when squared is 0.79 and describes the proportion of variance in common between the number of 802.15.4 failures and received 802.11 packets when averaged over a window of 26 s. This suggests that in this experiment 802.11 was indeed the cause for considerable packet loss, at least on channel 18.

When we repeated the measurement at other locations we observed less correlation (correlation coefficients around r = -0.4). It must be noted, however, that the correlation coefficients indicate only linear dependency and that the number of received 802.11 packets is a rather coarse metric because it does not take, for example, packet size or signal strength into consideration.

5 Evaluation of Selected Aspects

The measurement results from three different urban environments provided us with a large dataset to explore. In the following we analyze this dataset to determine (1) the interrelation of the transmission quality on different channels at a given point in time, and (2) whether trends in the noise floor can indicate a (future) decrease of transmission quality. The first helps understanding to what extent channel hopping could improve transmission quality, the second gives insight in a potential (dynamic) trigger for channel switching.

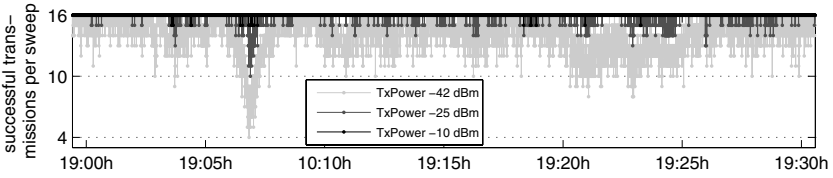


Fig. 8. Successful transmissions per sweep for the measurements described in Sect. 4.1

Our evaluation is an important starting point, but also has its limitations: it is trace-driven and therefore the results are applicable only for the particular environments, the specific mobility pattern and node placement. Our measurement setup also allows us to abstract from many practical issues such as scheduling periodic noise sampling so that transmissions of other nodes do not interfere.

5.1 Cross-Channel Quality Correlation

We are interested in the correlation of transmission failures over different channels at (roughly) the same time. Little correlation would mean that they share little variance and thus there is a greater probability that a "good" channel is available at a given point in time. In the following we first examine the fraction of good channels over time, then report on the empirical correlation between the channels and finally evaluate post factum how many hops an ideal frequency hopping scheme would have required to achieve 0 % transmission failures.

Fig. 8 shows the number of successful transmissions per sweep over time for the measurement described in Sect. 4.1 (it was the environment where we observed most transmission failures). At −10 dBm and −25 dBm transmission power for the majority of time the transmissions succeeded on almost all channels. Even for −42 dBm the number of good channels rarely dropped below 10, only at around 19:07 h temporarily 13 out of the 16 available channels were blocked. This means that in principle there were enough good channels at any point in time.

From the figure one cannot conclude how much variance the different channels have in common. We calculated the empirical correlation coefficients for the number of failed transmissions over time between all channels and found that they are often very low (typically around zero). For example, for −42 dBm transmission power the maximum correlation coefficients for any two channels in the three measurements were $r^2 = 0.1844$, $r^2 = 0.1052$, and $r^2 = 0.0336$, respectively.

We are interested in the minimum number of channel switches that would have been required to achieve 0 % transmission failures. In other words, the minimum number of hops that an ideal channel hopping scheme with precise knowledge of the future channel conditions would have taken to guarantee that all transmissions succeeded. We implemented a simple greedy algorithm that replays the empirical traces, starting from the first transmission and proceeding in time. The algorithm examines all transmissions and chooses the channel that provides 0 % transmission failures for the maximum time ahead. It proceeds

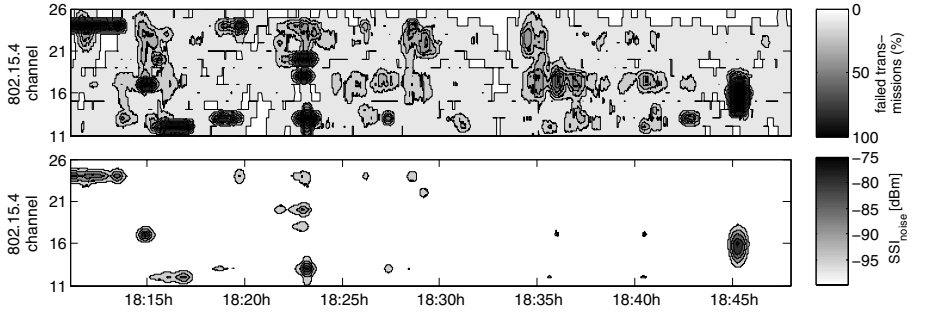


Fig. 9. Failed 802.15.4 transmissions at −42 dBm transmission power while walking through a central urban residential area (top). The bottom figure shows the noise floor measured during the same experiment. The results are averaged over a window of 100 transmissions per channel (26 s).

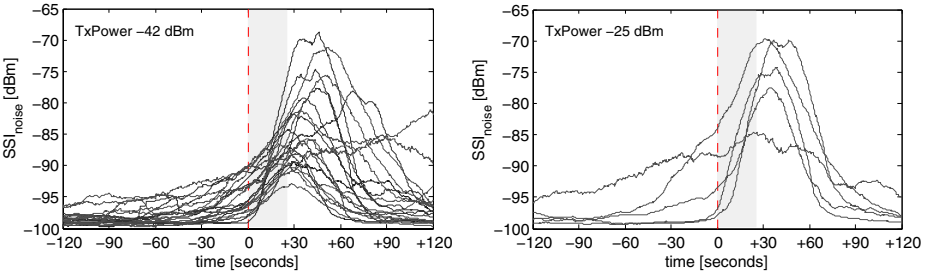
in time on this channel until a transmission failure occurs and switches the channel during this very sweep, again by choosing the channel that provides 0 % transmission failures for the maximum time ahead (from the previous evaluation we know that there is always a "good" channel). It repeats this step until it reaches the end of the empirical trace and then outputs the overall (minimum) number of channel switches (hops).

When the algorithm replayed the traces from the three measurements the total number of channel switches for −10 dBm transmission power was zero for all three empirical traces, which means there was at least one channel on which no failures occurred, respectively. At −25 dBm transmission power two or three hops were required. And at −42 dBm the measurement described in Sect. 4.1 required 50 hops (note that Fig. 5 shows moving averages over 100 transmissions), resulting in an average hopping frequency of about 2 Hz. The empirical results from the residential and office area corresponded to 27 and 38 channel switches, respectively.

5.2 Prediction of the Link Quality Degradation

A brief examination of the measurement results revealed that a substantial increase in noise floor is typically accompanied by heavy packet loss, in particular at −42 dBm transmission power. For example, Fig. 9 shows failed transmissions over time (top) together with a contour plot of the noise floor (bottom) averaged over 100 transmissions; this measurement was made in an urban residential area on a Friday afternoon.

We are interested in the dynamics of SSI_{noise} around the point in time when the quality of a communication link experiences significant degradation. For −42 dBm transmission power we define the threshold as ≥ 90 % transmission failures within 100 transmissions on a given channel, that means ≥ 90 % failures within 26 s. An analysis of the empirical traces reveals 8 (shopping street), 11 (residential area) and 2 (office area) = a total of 21 occurrences of significant



(a) −42 dBm transmission power: between 0 s and 26 s ≥ 90 % transmissions failed.

(b) −25 dBm transmission power: between 0 s and 26 s ≥ 50 % transmissions failed.

Fig. 10. Noise floor within a window of ±2 minutes around heavy transmission failures (starting at 0s). SSI_{noise} is arithmetically averaged over the past 26 s (prior moving average).

link quality degradation. For every occurrence we extract SSI_{noise} on the given channel within a window of ±2 minutes (roughly 1000 noise floor samples and corresponding to a SSI_{noise} sampling frequency of 4 Hz). Note that in our analysis we count only the first occurrence per channel (otherwise there was a total of 26 occurrences) and we ignore channels that already had significant losses at the beginning of the measurement, because for them the previous 2 minutes are not available.

Fig. 10a shows the results for −42 dBm transmission power. Since we are interested in general trends independent of the environment or a particular channel it includes the results from all measurements, one graph for every occurrence of link quality degradation. Each graph is aligned relative to the *beginning* of significant link quality degradation, which is represented by the vertical dashed line at time $t = 0$ s. This means, starting from $t = 0$ s the next 90 or more out of 100 transmissions failed, respectively. SSI_{noise} is arithmetically averaged over the *past* 26 s (prior moving average). A single point on one of the 21 graphs thus includes the current as well as the history of 99 previous noise floor samples.

At −25 dBm output power transmission failures were less frequent and we therefore reduced the threshold to ≥ 50 % transmission failures within a window of 26 s. This resulted in a total of 5 occurrences of link quality degradation as shown in Fig. 10b.

Discussion. The graphs show similar trends: at time $t = -120$ s the average (past) noise floor is typically below −95 dBm, respectively. Between $t = -120$ s and $t = 0$ s it increases slightly and the most substantial increases are observable around and especially short after $t = 0$ s, which is also the *start* of significant transmission failures. Right-shifting the dashed line by 13 s as well as left-shifting the noise floor graphs by 13 s, respectively, establishes temporal alignment between the moving averages. It is then clearly observable that significant link quality degradation corresponds with a *simultaneous* increase in average noise floor, which strongly suggests that external RF interference is indeed causing the

packet loss. The graphs are typically bell-shaped, which is likely a result from walking past a stationary interferer (for example, a WLAN access point).

The results suggest also that (the history of) noise floor observations may be valuable input to a link estimator. Especially at lower transmission frequency trends in the noise floor may be observable before a communication link experiences significant degradation. Increasing the noise floor sampling frequency and using more elaborated statistical techniques than a simple moving average are likely to have better predictive quality. We consider these topics part of our future work.

6 Related Work

The problem of coexistence between IEEE 802.11 and IEEE 802.15.4 networks has received significant interest from the research community. Most early work concentrated on developing probabilistic models that capture the dependence of interference-related packet loss in a 802.15.4 network based on frequency overlap and duty cycle, transmit power and distance of an 802.11 interferer [18]. Others analyzed the reverse problem, that is the impact of 802.15.4 networks on 802.11 devices [7], concluding that it is little to non-existing. A recent experimental study comes to a different conclusion, reporting that 802.15.4 devices may cause significant packet loss in an 802.11 network under specific conditions [16]. Prior work assessing the impact of WLAN interference on static 802.15.4 networks in lab environments typically reported on severe packet loss at small distances between the interfering devices [5].

Recently several 802.15.4 radio chip manufacturers have published guidelines to mitigate interference effects between the two technologies [11,17,4], for example, through minimal frequency offset of 20 MHz , spatial separation of 2 m and the use of the complete protocol stack (using ARQ to translate losses into latency) [17]. Acknowledging the problem, the IEEE 802.15 Task Group 4e currently investigates how to incorporate frequency hopping in the MAC layer. Meanwhile, recent revisions of standards that build on top of the 802.15.4, already incorporate simple frequency agility methods like periodic random channel hopping [19,6].

There is not so much experimental work on the specific challenges and opportunities of 802.15.4 BANs. Some recent studies have examined the performance of mobile 802.15.4 person-to-person communication, as well as with static receivers [3,12]. This work targets the impact of the human body on an inter-BAN communication link under specific mobility patterns, rather than external RF interference. Despite their static setup, the study presented in [14] is closest to our work: it focuses on detecting and mitigating the WLAN interference impact on 802.15.4 networks in an office setting. Targeting stationary networks, their measurement setup is optimized for more stable interference configurations, which is also reflected in the significantly higher duration of the sweep time compared to our setup (1.6 s vs. 85 ms). Their results confirm the correlation between 802.15.4 packet loss and 802.11 activity, as well as the suitability of noise-based predictors of WLAN interference.

7 Conclusions and Future Work

The effects that we observed in the isolated baseline measurements could, to some extent, also be recognized in the urban environments: transmission failures sometimes span over multiple consecutive 802.15.4 channels, are often correlated in time and substantial losses are typically accompanied by an increase in the noise floor. This suggests that external interference, in particular the virtually omnipresent WLAN, can be a major cause for substantial packet loss in IEEE 802.15.4 body area networks. However, in our configuration this is true only for the very low-power regime: already at −10 dBm transmit power transmission failures were negligible.

An obvious conclusion for the design of BAN protocols is to use higher transmission power (the IEEE 802.15.4 default is 0 dBm). On the other hand, there are several arguments for using low transmission power in BANs: less interference for other networks; less absorption of electromagnetic energy by the human body; less energy spent by the transceiver and thus longer lifetime (especially important for implanted sensors); less susceptibility for eavesdropping.

Adaptive transmission power control seems a promising approach to unite these requirements. Investigating the overhead of more effective interference evasion mechanisms, more intelligent noise "probing" approaches, combined with learning algorithms, as presented in [15], seem to be another promising direction of research that warrants experimental validation.

References

1. Yang, G.-Z. (ed.): Body Sensor Networks. Springer, Heidelberg (2006)
2. Texas Instruments. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver (April 2002), http://www.ti.com/lit/gpn/cc2420
3. Espina, J., Falc, T., Mülhens, O.: Network Topologies, Communication Protocols, and Standards. In: Body Sensor Networks, pp. 145–182. Springer, Heidelberg (2006)
4. Freescale Semiconductor. Mc1319x coexistence - application note, http://www.freescale.com
5. Golmie, N., Cypher, D., Rébala, O.: Performance analysis of low rate wireless technologies for medical applications. Computer Communications 28(10), 1255–1275 (2005)
6. HART field communication protocol specifications: TDMA data link layer specification. HCF_SPEC-75 (2008)
7. Howitt, I., Gutierrez, J.: IEEE 802.15.4 low rate - wireless personal area network coexistence issues. In: Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE, vol. 3, pp. 1481–1486 (2003)
8. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). IEEE Std 802.15.4-2003, pp. 1–670 (2003)

9. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. - part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs). IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002), pp. 1–580 (2005)
10. IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999), pp. C1–1184 (December 2007)
11. Jennic Ltd. Co-existence of IEEE 802.15.4 at 2.4 GHz - application note, http://www.jennic.com/
12. 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)
13. Moteiv Corporation. Tmote sky datasheet, http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf
14. Musaloiu-E, R., Terzis, A.: Minimising the effect of WiFi interference in 802.15.4 wireless sensor networks. Int. J. Sen. Netw. 3(1), 43–54 (2008)
15. Pollin, S., Ergen, M., Timmers, M., Dejonghe, A., van der Perre, L., Catthoor, F., Moerman, I., Bahai, A.: Distributed cognitive coexistence of 802.15.4 with 802.11. In: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 2006, June 2006, pp. 1–5 (2006)
16. Pollin, S., Tan, I., Hodge, B., Chun, C., Bahai, A.: Harmful coexistence between 802.15.4 and 802.11: A measurement-based study. In: 3rd International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 2008. CrownCom 2008, May 2008, pp. 1–6 (2008)
17. Thonet, G., Allard-Jacquin, P., Colle, P.: ZigBee - WiFi coexistence, white paper and test report. Technical report, Schneider Electric (2008)
18. Yoon, D.G., Shin, S.Y., Kwon, W.H., Park, H.S.: Packet error rate analysis of IEEE 802.15.4 under IEEE 802.11b interference. In: IEEE 63rd Vehicular Technology Conference, 2006. VTC 2006-Spring, May 2006, vol. 3, pp. 1186–1190 (2006)
19. ZigBee Alliance. ZigBee specification. ZigBee Document 053474r17 (2008)

Flow-Based Real-Time Communication in Multi-Channel Wireless Sensor Networks

Xiaodong Wang[1], Xiaorui Wang[1], Xing Fu[1], Guoliang Xing[2], and Nitish Jha[1]

[1] Department of EECS, University of Tennessee, Knoxville, TN 37996
[2] Department of CSE, Michigan State University, MI 48824
{xwang33,xwang,xfu1,njha}@utk.edu, glxing@cse.msu.edu

Abstract. As many radio chips used in today's sensor mote hardware can work at different frequencies, several multi-channel communication protocols have recently been proposed to improve network throughput and reduce packet loss for wireless sensor networks. However, existing work cannot utilize multiple channels to provide explicit guarantees for application-specified end-to-end communication delays, which are critical to many real-time applications such as surveillance and disaster response. In this paper, we propose MCRT, a multi-channel real-time communication protocol that features a flow-based channel allocation strategy. Because of the small number of orthogonal channels available in current mote hardware, MCRT allocates channels to network partitions formed based on many-to-one data flows. To achieve bounded end-to-end communication delay for every data flow, the channel allocation problem has been formulated as a constrained optimization problem and proved to be NP-complete. We then present the design of MCRT, which includes a channel allocation algorithm and a real-time packet forwarding strategy. Extensive simulation results based on a realistic radio model demonstrate that MCRT can effectively utilize multiple channels to reduce the number of deadlines missed in end-to-end communications. Our results also show that MCRT outperforms a state-of-the-art real-time protocol and two baseline multi-channel communication schemes.

1 Introduction

Many wireless sensor networks (WSN) applications heavily rely on information being transmitted in a timely manner. For example, a WSN-based disaster warning system must report detected events within a specified real-time deadline. Likewise, a surveillance system needs to notify authorities promptly upon the detection of any intruders. In WSNs, due to the lossy nature of wireless links, real-time communication protocols are commonly designed to provide only soft probabilistic real-time guarantees. There are many factors that may affect the end-to-end delay of a packet from the source to the destination (*e.g.*, a base station). Among them, a major factor is the retransmission caused by unreliable wireless links and channel contention [1].

A common way to improve link quality is to increase transmission power [2]. Transmission power can be used as a knob to reduce end-to-end delays due to

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 33–52, 2009.
© Springer-Verlag Berlin Heidelberg 2009

several advantages. First, it is supported by current sensor mote hardware. For example, the CC2420 radio chip [3] used in many mote hardware platforms has 31 different transmission power levels. Second, higher transmission power can lead to higher signal to noise ratio, hence reduce the number of retransmissions for a packet to be delivered [2]. Third, it may also increase the area range of high packet reception rate (*i.e.*, boundary of the gray area) of each node [1], and thus may lead to reduced number of hops needed to reach the destination. Previous work [4] has also shown that desired delays can be achieved by adapting transmission power of each node along an end-to-end path. However, a well-known drawback of increasing power for shorter delays is that high transmission power may significantly increase interferences and channel contention. As a result, the network capacity may be reduced [5]. This has greatly limited the feasibility of using transmission power to provide real-time guarantees.

Recently, multi-channel communication protocols have been proposed for WSNs to improve the performance of traditional single-channel protocols commonly used in WSNs. For example, a multi-channel protocol has been designed in [6] to improve network throughput and reduce packet loss for WSNs. Multi-channel MAC protocols [7,8] have also been proposed to improve network throughput for WSNs. Their simulation results show that those multi-channel protocols outperform their corresponding single-channel protocols. Multi-channel communications are promising because many radio chips used in today's sensor mote hardware can work at multiple frequencies. For example, the CC2420 radio chip provides 16 non-overlapping channels with radio frequency from 2,400 to 2,483MHz. However, existing multi-channel protocols do not provide explicit guarantees for application-specified end-to-end communication delays. On the other hand, as demonstrated in [9], multiple channels can significantly increase network capacity and thus greatly alleviate the drawback of using transmission power as a knob to achieve desired communication delays.

In this paper, we propose MCRT, a Multi-Channel Real-Time communication protocol that utilizes both multiple channels and transmission power adaptation for real-time WSNs. MCRT features a flow-based channel allocation strategy that is designed based on the multi-channel realities identified in a recent empirical study [6]. In particular, MCRT uses only a small number of orthogonal channels and avoids costly time synchronization. In MCRT, channels are allocated to network partitions formed based on many-to-one data flows in a WSN. To achieve bounded end-to-end communication delay for every data flow, we formulate the channel allocation problem as a constrained optimization problem and reduce it to the Maximum Length-Bounded Disjoint Paths problem [10], which is known as NP-complete. We then present an allocation algorithm designed based on well-established heuristics [11] to maximize the number of disjoint paths in the network that can meet the specified end-to-end communication delay. MCRT allocates a different channel to each data flow to minimize the channel contention among different flows. After the network partitions are established, transmission power adaptation is used to achieve energy-efficiency while forwarding each packet with real-time guarantees. We compare MCRT

against three baselines. The first baseline is a simple flow-based channel allocation solution. The second baseline has a node-level channel allocation strategy. The third baseline is a recently published work [4] that uses only transmission power to achieve real-time performance on a single channel. Extensive simulation results based on a realistic radio model demonstrate that MCRT outperforms all the three baselines and can effectively utilize multiple channels to reduce the number of deadlines missed in end-to-end communications. Specifically, the contributions of this paper are four-fold.

- We formulate the flow-based channel allocation problem in multi-channel real-time communications as a constrained optimization problem.
- We prove that the channel allocation problem is NP-complete and present a novel allocation strategy based on well-designed heuristics.
- We combine our channel allocation strategy with a power-efficient real-time packet forwarding protocol to achieve bounded communication delay on each channel.
- We evaluate the performance of our protocol against three baselines using extensive simulations in ns-2.

The rest of paper is organized as follows. Section 2 highlights the distinction of our work by discussing the related work. Section 3 introduces the formulation, proof, and algorithm of our flow-based channel allocation strategy. Section 4 discusses power-efficient real-time packet forwarding. Section 5 provides the design of the baselines used in our experiments. In Section 6, we evaluate the performance of our protocol using simulations. Section 7 concludes the paper.

2 Related Work

Many real-time communication protocols have been proposed for wireless sensor and ad hoc networks. A comprehensive review of real-time communication in WSNs is presented in [12]. At the MAC layer, Implicit EDF [13] is a collision-free real-time scheduling scheme by exploiting the periodicity of WSN traffic. RAP [14] uses a novel velocity monotonic scheduling scheme to prioritize real-time traffic based on a packet's deadline and distance to the destination. At higher layers, SPEED [15] achieves desired end-to-end communication delays by enforcing a uniform communication speed throughout the network. MMSPEED [16] can provide QoS differentiation to meet both reliability and timeliness requirements. SWAN [17] also proposes stateless control algorithms for differentiated services. Karenos et al. [18] have also presented a flow-based real-time traffic management mechanism. However, none of the existing real-time protocols takes advantage of the capability of multi-channel communications available in today's mote hardware. Our proposed MCRT protocol is specially designed for real-time communications in multi-channel WSNs.

Recently, several multi-channel MAC protocols have been proposed for WSNs [7] [8]. In these protocols, channels are assigned to different nodes locally to minimize interferences. This strategy is referred to as node-based channel assignment.

In node-based protocols, a node usually has a different channel from its downstream node and upstream node in a data flow. Therefore, each pair of nodes must switch to the same channel for communication, which may require precise time synchronization and lead to non-trivial overhead. In addition, some node-based strategies may require a large number of orthogonal channels, which may not be practical for existing mote hardware, as discussed in [6]. Nonetheless, simulation results demonstrate that these protocols can improve communication performance such as network throughput for WSNs. In this paper, one of our baselines also uses node-based channel assignment but requires neither time synchronization nor a large number of orthogonal channels. In contrast to the above related work, MCRT is designed to achieve application-specified end-to-end delays by using only a small number of orthogonal channels.

Some recent work [6] [19] proposes coarse-grained channel assignment policies, which allocate channels to disjoint trees or partitions. By minimizing the interferences between different trees or partitions, parallel transmissions can be exploited. In addition, experiments on Micaz hardware are also presented in [6] to investigate multi-channel realities. Two important realities have been reported. First, the number of orthogonal channels is actually small such that a practical multi-channel protocol should rely on only a small number of non-adjacent channels. Second, time synchronization protocols in WSNs could be expensive, in terms of bandwidth and power consumption. Hence, frequent re-synchronization should be avoided in protocols design. Our proposed MCRT protocol organizes the network into different partitions based on data flows, such that the interferences among different flows can be minimized. In addition, MCRT is designed to achieve desired end-to-end communication delays and reduce power consumption at the same time, which are not addressed in existing multi-channel work.

Transmission power control for energy efficiency has been studied extensively in the context of wireless ad hoc networks. The previous work can be roughly classified into two categories: topology control and power-aware routing. Topology control preserves the desirable property of a wireless network (e.g., connectivity) by reducing transmission power to the maximum degree. A survey on existing topology control schemes can be founded in [20] and several representative projects are [21] [22] [23] [24]. The goal of power-aware routing is to find energy-efficient routes by varying transmission power, as presented in [25] [26] [27] [28] [29]. Although the above studies demonstrate the effectiveness of transmission power control in reducing energy consumption, none of them deals with real-time requirements in multi-channel WSNs. In our work, we propose multi-channel protocol that uses transmission power adaptation to meet packet deadlines.

Different from all the aforementioned work that handles real-time guarantees, multi-channel communications, and energy-efficiency in isolation, our MCRT communication protocol utilizes the realistic capabilities of existing sensor mote hardware to support power-efficient multi-channel communications for real-time WSNs.

3 Flow-Based Channel Allocation

Our MCRT protocol features a flow-based channel allocation policy, which is mainly motivated by two observations. First, we conducted hardware experiments to motivate this work. As shown in our empirical results presented in an extended version of this paper [30], multiple data flows in a WSN compete for the shared wireless channel and thus result in degraded real-time performance. Hence, it is preferable that each data flow uses a different channel. Second, dynamic channel switching at the node level incurs overhead in terms of switching delay and energy consumption. Therefore, it is also preferable that nodes do not need to switch channel too frequently for data transmissions in a data flow.

In our flow-based channel allocation policy, we try to allocate a different channel to each data flow in the network such that the interferences among different data flows can be reduced. A data flow is composed of a source node and the destination, as well as the intermediate nodes in the network that can be used to forward packets from the source node to the destination. Data packets are periodically sent from the source node to the destination in a data flow, but each data packet may take a different path in the flow under different network conditions. Since each data flow is using a different channel, any node in the network (except the destination) can only be allocated to at most one data flow. To establish data flows, we partition the network by searching for a set of disjoint paths from source nodes to the destination. In order for each data flow to meet the specified deadline, the worst-case end-to-end communication delay of each path needs to be smaller than the deadline.

In this section, we first formulate the problem of finding disjoint paths with bounded delay as a constrained optimization problem. We then prove that the problem is NP-complete and propose a search algorithm based on well-established heuristics [11] to find the required number of disjoint paths in the network.

3.1 Problem Formulation

As discussed in [31], any two nodes A and B in a WSN may have three types of communication relationship. First, if B can reliably receive packets transmitted by A, we say that there exists a communication link from A to B. Second, if B cannot reliably receive packets from A, but A's transmission interferes with any transmission intended for B, we say that there exists an interference link from A to B. Last, if A and B cannot communicate or interfere with each other, there is no link between A and B. In this paper, we use *Packet Reception Rate (PRR)* to determine the communication relationship between two nodes in the WSN. Based on the empirical studies presented in previous work [24], we set a link with $PRR \geq 90\%$ to be a communication link and a link with $PRR \geq 10\%$ to be an interference link. The PRR value of each link can be measured using an online link quality estimator (*e.g.*, [32]).

Based on the definitions of communication and interference links, a WSN can be represented as a directional graph G = (V, E), where V is a set of nodes

and E is a set of directional links. In the graph, each link is assumed to be uni-directional. Our assumption is reasonable in our applications because we try to meet the end-to-end deadline for the data flows from the source nodes to the destination. In our real-time forwarding algorithm presented in Section 4, we adopt a greedy forwarding policy, *i.e.*, every node forwards packets to neighbors that are closer to the destination. As a result, based on the locations of the nodes, all the communication links in the network can be treated uni-directional. In this paper, we assume that each node is stationary and knows its location via GPS or other localization services. This assumption is reasonable because localization is a basic service essential to many WSN applications that need to know the physical location of sensor readings.

In order for each data flow to meet the given end-to-end communication deadline, we first consider the one-hop delay for a node to successfully receive a packet from another node along a data flow. Without accounting for interferences, the expected number of re-transmissions needed for the node to successfully receive a data packet from the other node can be calculated as the inverse of the PRR value of the communication link between the two nodes. The retransmission number is initially used as the weight of the communication link. For example, the weight of the link from E to B in Figure 1(a) can be calculated as 4, which corresponds to a PRR of 25%. We then consider the interferences that may increase the one-hop delay from E to B. Because interferences occur at the receiver, the communication link FB and the interference link AB in Figure 1 may interfere with EB. The longest delay that EB could experience is when all the transmissions happen to occur at the same time, *i.e.*, E is sending a packet to B; F is also sending a packet to B; A is interfering with B by sending a packet to another node (*e.g.*, C). The worst-case for EB is that E has to wait for both F and A to finish their transmissions before starting its own transmission. Therefore, the worst-case delay for EB is the aggregate weight of all the communication and interference links directed to B. As a result, the one-hop delay of EB is estimated as 12 (retransmissions). The weight of the interference link AB can be estimated as the maximum weight of the outgoing communication link from A, because EB needs to wait for the longest time when A is sending a packet to C. By doing this calculation for every communication link, we have a new graph where the weight of each link is its longest one-hop delay, as shown in Figure 1(b).

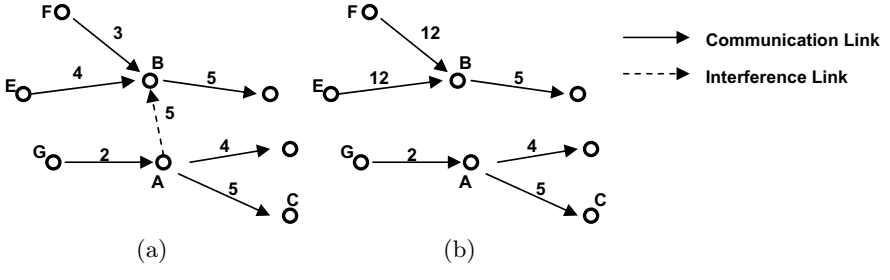


Fig. 1. An example of estimating the worst-case one-hop delay

With the one-hop delay of each link, we now need to find a set of disjoint paths from the source nodes to the destination such that the end-to-end delay of each path is smaller than the deadline. The delay of a path is calculated as the sum of the weights of all the links in the end-to-end path. Those paths are used to partition the network to form data flows with bounded communication delays. Therefore, the problem of finding Disjoint Paths with Bounded Delay (DPBD) can be formulated as a constrained optimization problem as follows. Given a directed graph G = (V, E) with k source vertices as $s_1, \ldots, s_k$, a destination vertex t, and a set of edges with various weights, we need to find k or more mutually vertex-disjoint (except the sources and destination) paths from $s_i, (1 \leq i \leq k)$ to t. The optimization problem is subject to the constraint that the weight (*i.e.*, delay) of each path needs to be smaller than the deadline W. If the number of vertex-disjoint paths is greater than k, some data flows can have more than one path. If we cannot find a path from a source to the destination, the end-to-end delay of that data flow cannot be guaranteed to be smaller than the deadline.

3.2 Proof of NP-Completeness

We now prove the DPBD problem to be NP-complete by reducing it to a well-known NP-complete problem, the Maximum Length-Bounded Disjoint Paths (MLBDP) problem [10], which is stated as follows. Given a graph G = (V, E) with specified vertices s, t and positive integers k, $W' \leq \|V\|$, does G contain k or more mutually vertex-disjoint paths from s to t, none involving more than W' edges?

Theorem 1. *The DPBD problem is NP-Complete.*

Proof. First, it is clear that our problem belongs to NP problem because given a set of k disjoint paths, we can check if the weight of each path is bounded by the value W. This check can be performed in polynomial time.

We now reduce our problem to the MLBDP problem. There are two differences between our DPBD problem and the MLBDP problem. First, the edges (*i.e.*, links) in our graph have various weights while the edges in the MLBDP problem have a uniform weight of 1. Second, we need to find one or more paths from each of the k source nodes to the same destination. However, the MLBDP problem aims to find k or more paths between the same source s and the same destination t. We use two steps to reduce our problem to the MLBDP problem.

In the first step, as the weight of each edge is a rational number, we can always find the greatest common denominator for all the edge weights in the graph, which is denoted as c. Thus, the weight of each edge can be expressed as $I \times 1/c$, where I is an integer. We then replace this edge with a chain composed of I new edges (with a weight of $1/c$) and $I - 1$ new intermediate vertices, as shown in Figure 2(a). As a result, the total weight between the two vertices of the original edge is still $I \times 1/c$ while each edge now has a uniform weight of $1/c$. All the edges in the graph can be replaced in the same way, which leads to a new graph where all the edges have the same weight. In the second step, we

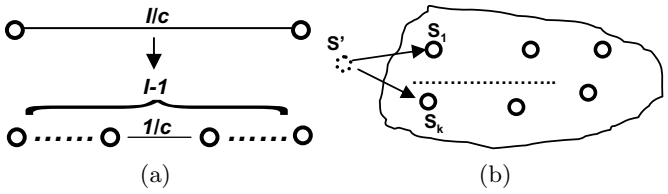


Fig. 2. Graph transformation of the DPBD problem

first add an auxiliary vertex, denoted as s' to the graph G. We then link s' to each of the k source vertices with an edge whose weight is the uniform value, as shown in Figure 2(b). If we can find k disjoint paths from s' to t, each source node will have one path to the destination with bounded delay.

After the two steps, we have transformed our graph to a new graph G' = (V',E') with specified vertices s', t and positive integers k, $W' = W \times c + 1$. The DPBD problem is reduced to a new problem stated as follows. Given the new graph G', does G' contain k mutually vertex-disjoint paths from s' to t, none involving more than W' edges? The new problem is exactly the MLBDP problem. Therefore, our problem is NP-complete. □

3.3 Disjoint Paths Search Algorithm

In this section, we propose a search algorithm designed based on well-established heuristics [11] to find the required number of disjoint paths in the new graph G' in two steps.

In the first step, the algorithm adopts the Dijkstra's algorithm to find the shortest path from s' to the destination t in the network. If the shortest path is not bounded by W', it is impossible to find k disjoint weight-bounded paths. In that case, the search algorithm fails. If the shortest path is bounded, it is added to the solution set T.

In the second step, based on the shortest path found in the first step, the algorithm iteratively search for the rest $k-1$ bounded disjoint paths. Every iteration of the algorithm finds a new path, whose length is bounded by W', and guarantees that all the paths found so far are disjoint. Note that each iteration may modify the paths found in previous iterations to maximize the number of bounded paths. Specifically, each iteration works as follows.

Starting from s', the algorithm adopts the Depth-First-Search (DFS) method to search for a new path toward t whose length is bounded by W'. Suppose that the search has reached node n and is looking for the next hop, as shown in Figure 3. In order to guarantee that the path found by DFS is disjoint from the existing paths in T, the algorithm first tries to pick the next-hop node of the new path from the neighbors of n that do not belong to any existing paths (referred to as free neighbors). If such a neighbor is available and the total length of the path after adding this neighbor is still smaller than the bound, the neighbor is picked by DFS as the next hop in the new path.

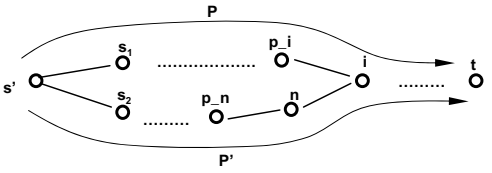


Fig. 3. Example of the Matching Procedure

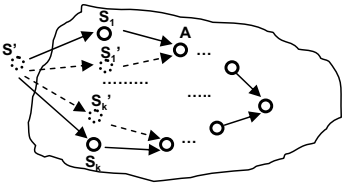


Fig. 4. Extended DPBD problem

If such a neighbor is unavailable, the algorithm starts an augmentation procedure called *matching*. The procedure checks if n has any neighbor, which belongs to a path in T, can provide a W' bounded path toward t. For example, suppose i is such a neighbor and i belongs to an existing path P in T. The procedure forms a new path P', which includes the current search path from s' to n, the link between n and i, and the part of path P from i to t. If the length of the new path is bounded by W', P is deleted from the solution set T and P' is added to T. The procedure then uses i's predecessor, p_i, in path P as the current node. After the matching procedure, the algorithm starts DFS again from node p_i.

Since DFS may fail to find the next hop and need to back off, the search may go back to node s'. In that case, if all the neighbors of s' have already been visited, it indicates that the last matching procedure was not successful. The algorithm then adds path P, which was deleted in the last matching procedure, back to T, and then removes the new path P' established in last matching from T. The algorithm then rolls back to continue DFS from node p_n, which is the predecessor of the current node n in the last matching procedure.

The whole algorithm terminates under two conditions. First, if the destination t is reached, the algorithm has successfully found a new disjoint length-bounded path. Second, if the search goes back to s' with no more neighbor to visit and all the matching procedures conducted before have been rolled back, the algorithm fails to find a new disjoint length-bounded path. The number of paths in T is the maximum number of disjoint paths with bounded length that the algorithm can find. The detailed algorithm of finding a new disjoint path in the second step is presented in the pseudo code (Algorithm 1).

Based on the analysis in [11], the time complexity for DFS to find a new path is $O(W'\|E\|)$. The time complexity of the matching procedure in the algorithm is $O(W'^2\|V\|\|E\|)$. Therefore, the time complexity of finding a new disjoint path with bounded delay is $O(W'^2\|V\|\|E\|)$. The algorithm is currently a centralized procedure but will be extended to a distributed procedure in our future work.

In a real WSN, it is preferable to have multiple paths in each data flow for two reasons. First, a data flow composed of only a single path is not fault-tolerant as node failures may disconnect the flow. Second, with more nodes in a data flow, each node can have more choices to pick the most power-efficient next hop for packet forwarding while meeting the deadline requirement. Power-efficient real-time forwarding is discussed in detail in Section 4. Therefore, in our real implementation of the algorithm, we extend the DPBD problem to allow a

Algorithm 1. Finding One New Disjoint W' Bounded Path

```
Assume we have a solution set T that contains l ≤ k disjoint paths;
n ⇐ s'; Matching Stack Height ⇐ 0;
while n ≠ t do
  Use DFS to find next free neighbor n + 1 that provides W' length bounded path
  to t;
  if no free neighbor available then
    Find a neighbor i in path P ⊂ T, which can provide a W' length bounded path
    to t; Establish path P' through n and i; Push n into the stack; T = T − P + P';
    n ⇐ p_i;
    Continue;
  end if
  if no non-free neighbor available then
    n ⇐ p_n;
    if n = s' then
      if matching stack hight = 0 then
        Return failure.
      else
        n ⇐ stack pop out; T = T − P' + P;
      end if
    end if
  end if
end while
```

source node to have multiple (*e.g.*, m) paths. To this end, we further transform the new graph G' = (V',E') in the proof of Theorem 1 to make $m-1$ copies of each source node s_i, as shown in Figure 4. Each copy of a source node has the same edges that the source node has. For example, s_1's copy, s'_1, also has edges to s' and A. We then run the search algorithm to find the maximum number of disjoint paths for the new graph. As a result, some of the data flows may have multiple paths to the destination. After all the disjoint paths are found, the channel allocation algorithm merges all the copies for each source node in the graph. All the paths that share the same source node belong to the same data flow and are thus allocated a different channel. As a result, interferences among different data flows can be minimized.

4 Power-Efficient Real-Time Routing

Based on the theoretical analysis presented in Section 3, all data flows are guaranteed to have bounded end-to-end delay even when every node experiences the worst-case one-hop delay. In a real WSN, end-to-end deadline can be set to be shorter than the theoretical bound. In this section, we present the power-efficient real-time routing strategy adopted by MCRT, which is the second step of MCRT after the channel allocation step. Since each data flow is allocated a different channel, we consider the communication of one data flow in this section.

The design principle of our power-efficient real-time routing strategy is to use adaptive transmission power control to achieve required one-hop delay. Empirical results [2] demonstrate that higher transmission power may lead to improved link quality due to the increased packet reception rate. High reception rate will in turn reduce the number of retransmissions needed to deliver a packet, and thus reduce the transmission delay. Another advantage of power adaptation is energy efficiency. An unnecessary high power level may lead to excessive power consumption. In addition, high transmission power may cause increased interferences and channel contention, and hence reduce the network capacity. In this paper, we implement power adaptation to use just enough power for desired transmission delays. Please note that though we only control transmission power in this paper, our protocol can be integrated with energy-efficient WSN MAC protocols with periodic sleeping (*e.g.*, B-MAC [33]) for further energy saving at the cost of longer communication delays. The integration is our future work.

4.1 Real-Time Forwarding

Based on the single-channel real-time routing algorithm presented in [4], we adopt a dynamic velocity assignment policy and a forwarding policy based on delay estimation. We assume that a data flow periodically sends a packet to the destination. The end-to-end deadline of the data flow is embedded in the packet. Each node that receives the data packet needs to forward the packet to a neighbor based on whether the neighbor can meet the delay requirement of the packet at the *minimum cost of energy consumption*. In this paper, we use two metrics: *required velocity* and *provided velocity* to map a packet's end-to-end deadline to a set of local deadlines for each node to meet. Specifically, when a node needs to forward a packet, it calculates its local deadline, *i.e.*, the required velocity to be achieved for the current hop based on the following equation:

$$velocity_{required}(s,d) = \frac{dis(s,d)}{slack} \tag{1}$$

where $dis(s,d)$ is the Euclidean distance from the current node s to the destination node d. *slack* is the amount of time left before the deadline. Note that with this deadline assignment policy, if a packet can meet its required velocity at every hop, it can guarantee to meet its end-to-end deadline. The required velocity is recomputed at each hop. The slack is initially set to be the end-to-end deadline at the source node. At each hop, the slack is decremented to account for queuing, contention and transmission delays based on the estimation methods introduced in [4].

To meet the velocity requirement, the velocity that can be provided by each forwarding choice (*i.e.*, a neighbor node with a certain power level) in the neighborhood table is computed. In the case when node s forwards a packet to destination d using a forwarding choice (n,p,c), which means node n is selected as the next hop, p is the transmission power, and c is n's channel, the velocity provided by the forwarding choice is:

$$velocity_{provided}(n,p,c) = \frac{dis(s,d) - dis(n,d)}{delay(n,p,c)} \quad (2)$$

The one-hop delay $delay(n,p,c)$ is estimated based on the methods described in [4]. $dis(s,d) - dis(n,d)$ is the progress made toward the destination by forwarding the packet to node n. To ensure that the neighbor receives the packet, the node has to receive the MAC-layer ACK from the neighbor. If the number of needed retransmissions is larger than 5, the data packet is dropped. This multi-channel forwarding policy eliminates the need of costly time synchronization used in previous node-based multi-channel work (*e.g.*, [7] [8]).

4.2 Neighborhood Management

We adopt the reliable routing framework proposed in [32] to deal with the dynamic and lossy nature of WSNs. First, link quality and status need to be measured dynamically through a link estimator. Second, measured link quality must be maintained in a neighborhood table for making reliable routing decisions in dynamic environments. In our protocols, we measure the one-hop delay between the node and its each neighbor using data packets to avoid the overhead of probing packets. The delay information of each neighbor is stored in a neighbor table and used to make reliable routing decisions in our protocol. Specifically, we maintain a neighbor table for each node to record the provided velocity of each neighbor. When a node receives a data packet, it searches the table to find a neighbor that can provide the requested velocity and has the lowest transmission power. In that way, we use just enough power for the desired velocity and thus can achieve power-efficiency.

If no neighbor can provide the requested velocity, the node will select some neighbors to conduct power adaptation [2]. The neighbor node used in the last successful packet delivery to the same destination will be considered first, because its link status is most up-to-date. If the last used node is not eligible, the second last used node will be considered. We only consider those neighbor nodes that have a non-zero retransmission number as there is space for power adaptation to reduce their delays. If the neighbor's corresponding transmission power is not the highest power level yet, we use a policy similar to the well-known Multiple Increase Linear Decrease (MILD) backoff algorithm to adjust the power level used to transmit a packet to the neighbor. Specifically, the power level will be multiplied by 1.5 for timely delivery of the current data packet. For example, if the current power level is 10, a power level of 15 will be used to transmit the data packet. This policy is used because timeliness is regarded more important than energy-efficiency in this work. After the packet is successfully transmitted, the power level will be decreased by 1 and will continue to decrease upon every successful packet transmission to this neighbor.

If a node cannot find a neighbor eligible for power adaptation, it sends out a *Routing Request* (RR) packet to find new neighbors that can provide the required velocity. The RR packet contains the required velocity and neighborhood table information, and is broadcast using the highest power level. When neighbors

that are not currently in the neighbor table receive the RR packet, they check whether they can provide the required velocity. If a neighbor can provide required velocity, it replies to the RR packet after a random delay. When other neighbors overhear the reply, they stop sending replies to the current node to reduce the chance of network congestion caused by a large number of replies.

5 Design of Baseline Algorithms

In all our experiments, we compare our MCRT protocol against three well-designed baselines: a simple flow-based multi-channel real-time protocol, a node-based multi-channel real-time protocol, and a single-channel power-aware real-time protocol [4].

The first baseline we use to compare with MCRT is a simple flow-based multi-channel real-time protocol called SIMPLE, which is designed in the same way as MCRT except that it uses a simple heuristic to find disjoint paths for channel allocation. During the initialization phase of the network, the source node of each flow broadcasts an explorer packet on the common channel with the distance from the source to the destination attached. Nodes that receive this packet check its own distance to the destination. If its distance is shorter than that in the packet, it waits for a random time and then replies to the source node. Other nodes that overhear the reply will stop sending reply message to avoid network congestion. The packet is then forwarded to the first replying node. The process continues until the explorer packet arrives at the destination. A path from the source to the destination is then established. A multi-hop ACK packet is then sent from the destination back to the source. Every node on the path switches to the new channel immediately after successfully receiving the MAC-layer ACK. In our experiments, two explore packets are used to find two paths for a data flow.

The second baseline is a node-based multi-channel real-time protocol. In this protocol, instead of allocating channels to data flows, every node has its own default channel and needs to dynamically switch channel in order to communicate with another node. In the initialization phase of a WSN, every node claims its own default channel in a way to have approximately even distribution of the channels. During the data transmission phase, if the current node wants to forward a packet to a neighbor on a different channel, the node needs to switch to that channel to send the packet. To ensure that the neighbor receives the packet, the node has to receive the MAC-layer ACK from the neighbor before switching back to its own channel. Note that our node-based baseline eliminates the need of costly time synchronization used in previous node-based multi-channel work (*e.g.*, [7,8]). Different from the single-channel work such as [4], RR packets are broadcast on different channels. In our baseline, a node first broadcasts on its own channel and then switches to other channels to broadcast the RR packet. After that the node switches back to its own channel. If a qualified node needs to send a RR reply to the current node, it switches to the current node's channel to do so.

The last baseline is a single-channel real-time routing protocol called RPAR [4]. We compare MCRT against RPAR to show that multiple channels can be effectively utilized to reduce packet drop ratio, and thus reduce the number of needed retransmissions and communication delays, especially when the specified deadlines are tight. Note that RPAR outperforms several existing real-time and energy-efficient protocols (including one similar to SPEED [15]), by achieving a smaller deadline miss ratio and less energy consumption, as demonstrated in [4]. Therefore, by having better real-time performance and less energy consumption than RPAR, our MCRT protocol also outperforms the baseline protocols used by RPAR.

6 Performance Evaluation

In this section, we first introduce our simulation setup. We then present the simulation results to compare our MCRT protocol against the three baselines, under different transmission deadlines, data rates, number of data flows, and network densities.

6.1 Simulation Setup

We implement the MCRT protocol in the ns-2.29.3 release of the ns-2 network simulator [34]. We configure ns-2 based on the characteristics of Mica2 sensor motes. Each node has 31 transmission power levels, from -20 dBm to 10 dBm. The bandwidth is set as 40Kbps for the experiments. The probabilistic radio model in [35] has been implemented in ns-2 to model lossy links. The ns-2 simulator is also modified to support multiple channels and to allow dynamic channel switching. The MAC protocol used in our simulations is a simple CSMA scheme similar to B-MAC [33], the default MAC protocol in TinyOS.

The network topology used in the experiments includes 130 nodes distributed in a 150m $\times$ 150m area. The area is divided into 13 $\times$ 10 grids, each of which is roughly 13m $\times$ 10m. Each grid is configured to have a node randomly deployed in it. In Section 6.4, the network topology consists of 361 nodes distributed in the same area with a higher density. We use the common *many-to-one* traffic pattern in our simulations. In each experiment, the first source node is selected to be the node in the center of the left-most grid column in the network. Other source nodes are randomly selected from the left-most grids with a certain distance from each other. We assume the destination node (*i.e.*, the base station) is a special node that is equipped with multiple radio transceivers, such that it can receive packets on multiple channels simultaneously. We assume the destination locates just outside the right-side boundary of the network. The destination can directly talk, on different channels, to several adjacent nodes located in the center of the right-most grids. As long as a packet can be delivered from a source node to one of those nodes, the packet is assumed to be successfully delivered to the destination.

We use a traffic generator that varies the interval between two data packets based on the sum of a constant (300ms) and a random number generated by

an exponential distribution. The following setup is used in the experiments if not otherwise indicated. The network is configured to have 3 data flows from 3 source nodes to the destination. Each source node generates a new packet every 4 seconds. The end-to-end transmission deadline is 300ms. Three channels are used in our simulations due to the limited availability of orthogonal channels, as reported in [6]. All the nodes start with no neighbor information and thus have an empty neighborhood table.

We use two performance metrics to evaluate the performance of the four protocols: the MCRT protocol and the three baselines. The first metric is *deadline miss ratio* which is the fraction of data packets that miss their deadlines during end-to-end transmissions. This metric examines the real-time performance required in many real-time WSN applications. The second metric we use is *energy consumption per data packet*, which is the ratio between the total energy consumed in transmissions and the number of packets that successfully meet their deadlines. This metric evaluates the energy efficiency of the proposed protocols. Each data point in all the figures is the average of five different runs. The 90% confidence interval of each data point is also plotted.

6.2 Different Transmission Deadlines

The first set of experiments evaluates the performance of the four protocols under different end-to-end transmission deadlines. Figure 5 shows the deadline miss ratios when the deadline varies from 150 ms to 350 ms. MCRT has the lowest miss ratio among all the four protocols. MCRT has better performance than RPAR because it can utilize multiple channels for reduced communication delays. MCRT outperforms the node-based scheme because the node-based scheme needs to broadcast RR packet in multiple channels when it fails to find a neighbor that can provide the required velocity, which contributes to longer delay. The performance of MCRT is slightly better than that of SIMPLE because the data flows in MCRT are formed to be bounded even when every node has the worst-case one-hop delay. In contrast, as introduced in Section 5, SIMPLE randomly picks nodes to form data flows without considering interferences and one-hop delay of each node.

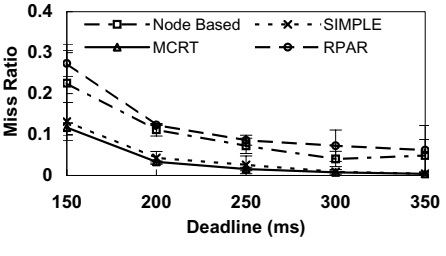


Fig. 5. Miss ratio when deadline is varied

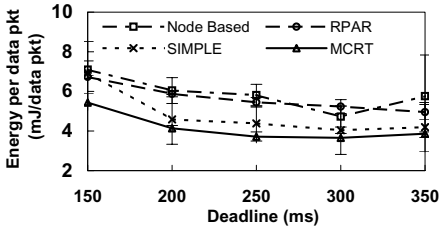


Fig. 6. Energy consumption when deadline is varied

Figure 6 shows the energy consumption of the four protocols. MCRT has the lowest energy consumption for all the deadlines. The reason is that MCRT has the smallest number of retransmissions, which greatly reduces the energy consumption. In addition, MCRT also has a much lower deadline miss ratio as shown in Figure 5. As a result, more packets successfully meet their deadlines, which leads to improved energy efficiency.

6.3 Different Data Rates

This set of experiments studies the performance of the four protocols when the data rate of the three source nodes is increased from one packet per 5 seconds to one packet every second. Figure 7 shows that there is no clear evidence that data rate may significantly affect the miss ratio for the four protocols. MCRT and SIMPLE have better real-time performance because they divide the network into partitions, with a different channel used in each partition. As a result, the interferences between different data flows can be reduced.

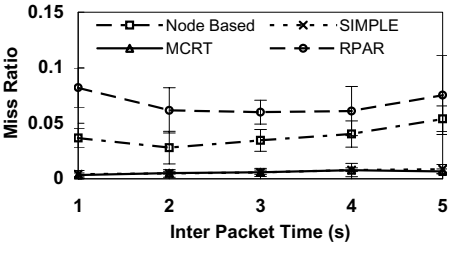


Fig. 7. Miss ratio when data rate is varied

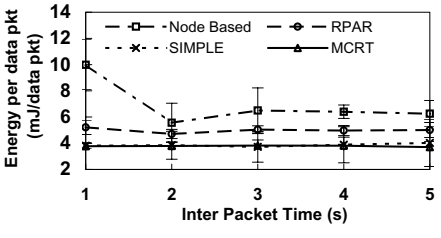


Fig. 8. Energy consumption when data rate is varied

Figure 8 shows the energy consumption. Both MCRT and SIMPLE have lower energy consumptions, compared with the other two protocols. This is because they have much fewer retransmissions caused by the channel contention among different data flows. The node-based scheme consumes the most energy because it needs to send out RR packets on multiple channels. In addition, it is easier for the node-based scheme to have some long-distance neighbors that may need higher power to successfully transmit packets. Those long-distance neighbors are due to the fact that the node-based scheme switches between multiple channels to broadcast RR packets, and thus has a smaller chance of finding nearby neighbors to provide the required velocity.

6.4 Different Number of Data Flows

In this experiment, we vary the number of data flows in the network from 2 to 6 and 361 nodes are deployed in the network. When the number of flows is greater than the number of channels, we try to evenly distribute the flows to each channel for the MCRT and SIMPLE protocols. For example, with four

flows and three channels, two of the flows will share a single channel. Figure 9 shows that the deadline miss ratios of MCRT and SIMPLE remain the same when the number of flows increases from 2 to 3. This is because each flow can transmit on a separate channel when the number of flows is smaller than or equal to 3. The miss ratios of MCRT and SIMPLE increase slightly when the number of flows increases from 3 to 6, because two flows need to share one channel, leading to slightly increased channel contention and deadline miss ratios. The increased number of flows has the biggest impact on RPAR, raising its deadline miss ratio to almost 30%. The results show that single-channel protocols are more vulnerable to the increasing number of competing data flows. On the other side, multi-channel protocols can utilize multiple channels to effectively reduce packet drop ratio, and so mitigate the impact of increased data flows.

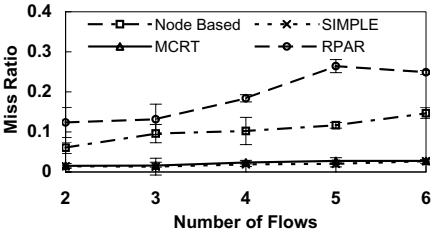


Fig. 9. Miss ratio when number of data flows is varied

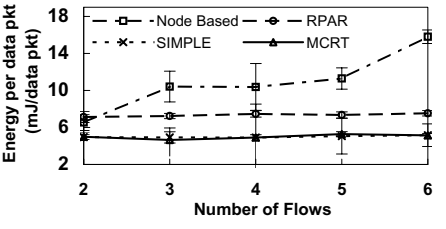


Fig. 10. Energy consumption when number of data flows is varied

As shown in Figure 10, MCRT and SIMPLE have lower energy consumption. The reason is that MCRT and SIMPLE have fewer retransmissions caused by the channel contention among different flows, as they have fewer flows in each network partition. MCRT is slightly better than SIMPLE (only except for 5 flows) because the delays of the data flows in MCRT are bounded, which results in fewer nodes in each flow and hence smaller number of transmissions to reach the destination. The node-based protocol has the highest energy consumption because each node is more likely to have long-distance neighbors, which require higher power for successful transmissions. In addition, the node-based protocol broadcasts RR packets in multiple channels, which contributes significantly to its energy consumption because more RR packets need to be sent when more data flows are competing for the channel.

6.5 Different Network Densities

In this set of experiments, we vary the network density by changing the spacing between every two nodes from 14m to 8m, which in turn changes the total number of nodes from 121 to 361. Figures 11 and 12 show the miss ratio and energy consumption for all the four protocols, respectively. The miss ratios of RPAR and the node-based protocol increase when the network density increases. This is because the neighborhood table of each node is filled up with some short-distance neighbors in a high density network. As a result, the required number of hops for

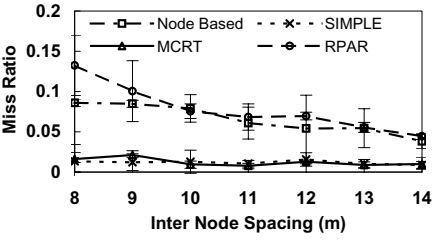


Fig. 11. Miss ratio when network density is varied

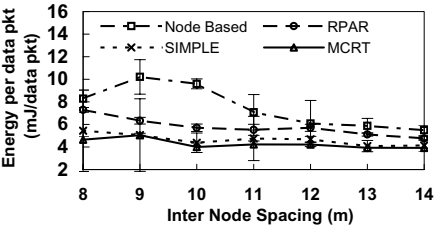


Fig. 12. Energy consumption when network density is varied

a packet to reach the destination is increased, making it hard to meet the deadline. MCRT and SIMPLE are not significantly impacted by the varying network density because partitioning the network leads to fewer short-distance neighbors in each data flow. The energy consumption decreases for all the four protocols when the network density decreases. This is because more packets are successfully transmitted to the destination due to smaller miss ratios and the number of hops to deliver a packet becomes smaller when the density is lower. MCRT has the lowest energy consumption because it has fewer retransmissions and a lower deadline miss ratio. The node-based protocol has the highest energy consumption because it uses more RR packets than other protocols.

7 Conclusion

In this paper, we have presented MCRT, a multi-channel real-time communication protocol that utilizes both multiple channels and transmission power adaptation to achieve real-time communications in WSNs. MCRT features a flow-based channel allocation strategy, which is designed based on the multi-channel realities identified in previous work to use only a small number of orthogonal channels. To achieve bounded end-to-end communication delay for every data flow, the channel allocation problem has been formulated as a constrained optimization problem and proved to be NP-complete. The design of MCRT includes a channel allocation algorithm designed based on well-established heuristics and a real-time packet forwarding strategy. Extensive simulation results demonstrate that MCRT can effectively utilize multiple channels to reduce the number of deadlines missed in end-to-end communications. Our results also show that MCRT outperforms a state-of-the-art real-time protocol and two baseline multi-channel communication schemes.

References

1. Zhao, J., Govindan, R.: Understanding packet delivery performance in dense wireless sensor networks. In: SenSys (2003)
2. Lin, S., He, T., Zhang, J., Zhou, G., Gu, L., Stankovic, J.A.: ATPC: Adaptive transmission power control for wireless sensor. In: SenSys (2005)

3. CC2420 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver, http://www.chipcon.com
4. Chipara, O., He, Z., Xing, G., Chen, Q., Wang, X., Lu, C., Stankovic, J., Abdelzaher, T.: Real-time power-aware routing in sensor networks. In: IWQoS (2006)
5. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on on Information Theory 46(2) (2000)
6. Wu, Y., Stankovic, J., He, T., Lin, S.: Realistic and efficient multi-channel communications in dense sensor networks. In: INFOCOM (2008)
7. Zhang, J., Zhou, G., Huang, C., Son, S.H., Stankovic, J.A.: TMMAC: An energy efficient multi-channel mac protocol for ad hoc networks. In: IEEE ICC (2007)
8. Zhou, G., Huang, C., Yan, T., He, T., Stankovic, J.A., Abdelzaher, T.F.: MMSN: Multi-frequency media access control for wireless sensor networks. In: INFOCOM (April 2006)
9. Kyasanur, P., Vaidya, N.H.: Capacity of multi-channel wireless networks: impact of number of channels and interfaces. In: MobiCom (2005)
10. Garey, M.R., Johnson, D.S.: Computer and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
11. Ronen, D., Perl, Y.: Heuristics for finding a maximum number of disjoint bounded paths. Networks 14, 531–544 (1984)
12. Stankovic, J.A., Abdelzaher, T., Lu, C., Sha, L., Hou, J.: Real-time communication and coordination in embedded sensor networks. Proceedings of the IEEE 91(7) (2003)
13. Caccamo, M., Zhang, L.Y., Sha, L.: An implicit prioritized access protocol for wireless sensor networks. In: RTSS (2002)
14. Lu, C., Blum, B.M., Abdelzaher, T.F., Stankovic, J.A., He, T.: RAP: A real-time communication architecture for large-scale wireless sensor networks. In: RTAS (2002)
15. He, T., Stankovic, J., Lu, C., Abdelzaher, T.: SPEED: A stateless protocol for real-time communication in sensor networks. In: ICDCS (2003)
16. Lee, E.F.C.-G., Ekici, E.: MMSPEED: Multi-path multi-speed protocol for qos guarantee of reliability and timeliness in wireless sensor networks. IEEE Transactions on Mobile Computing 5(6), 738–754 (2006)
17. Ahn, G.-S., Sun, L.-H., Veres, A., Campbell, A.T.: SWAN: Service differentiation in stateless wireless ad hoc networks. In: INFOCOM (2002)
18. Karenos, K., Kalogeraki, V.: Real-time traffic management in sensor networks. In: RTSS (2006)
19. Vedantham, R., Kakumanu, S., Lakshmanan, S., Sivakumar, R.: Component based channel assignment in single radio, multi-channel ad hoc networks. In: MobiCom (2006)
20. Santi, P.: Topology control in wireless ad hoc and sensor networks. Istituto di Informatica e Telematica, Pisa - Italy, Tech. Rep. IIT-TR-04 (2003)
21. Ramanathan, R., Hain, R.: Topology control of multihop wireless networks using transmit power adjustment. In: INFOCOM (2000)
22. Li, L., Halpern, J.Y., Bahl, P., Wang, Y.-M., Wattenhofer, R.: Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In: PODC (2001)
23. Li, N., Hou, J.C., Sha, L.: Design and analysis of an mst-based topology control algorithm. In: INFOCOM (2003)
24. Son, D., Krishnamachari, B., Heidemann, J.: Experimental study of the effects of transmission power control and blacklisting in wireless sensor networks. In: SECON (2004)

25. Singh, S., Woo, M., Raghavendra, C.S.: Power-aware routing in mobile ad hoc networks. In: MobiCom (1998)
26. Li, Q., Aslam, J., Rus, D.: Online power-aware routing in wireless ad-hoc networks. In: MobiCom (2001)
27. Chang, J.-H., Tassiulas, L.: Energy conserving routing in wireless ad-hoc networks. In: INFOCOM (2000)
28. Sankar, A., Liu, Z.: Maximum lifetime routing in wireless ad-hoc networks. In: INFOCOM (2004)
29. Doshi, S., Bhandare, S., Brown, T.X.: An on-demand minimum energy routing protocol for a wireless ad hoc network. SIGMOBILE Mob. Comput. Commun. Rev. 6(3) (2002)
30. Wang, X., Wang, X., Fu, X., Xing, G., Jha, N.: Flow-based real-time communication in multi-channel wireless sensor networks, Tech Report, University of Tennessee, Knoxville, TN (2008), http://www.ece.utk.edu/~xwang/papers/mcrt.pdf
31. Chipara, O., Lu, C., Stankovic, J.: Dynamic conflict-free query scheduling for wireless sensor networks. In: ICNP (2006)
32. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: SenSys (2003)
33. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys (2004)
34. ns-2 Network Simulator, http://nsnam.isi.edu/nsnam/index.php/Main_Page
35. Zuniga, M., Krishnamachari, B.: Analyzing the transitional region in low power wireless links. In: SECON (2004)

QoS Management for Wireless Sensor Networks with a Mobile Sink

Rob Hoes[1,2], Twan Basten[2,3], Wai-Leong Yeow[4], Chen-Khong Tham[1], Marc Geilen[2], and Henk Corporaal[2]

[1] National University of Singapore
[2] Eindhoven University of Technology
[3] Embedded Systems Institute
[4] Institute for Infocomm Research

Abstract. The problem of configuration of Wireless Sensor Networks is an interesting challenge. The objective is to find the settings, for each sensor node, that optimise certain task-level QoS metrics. An existing configuration method is available for tree-based static networks. We extend this method to support a mobile sink. First, the routing tree is adapted to the sink's new location, after which the parameters of the nodes are optimised. Both algorithms are distributed and localised, and therefore efficient and scalable, and are able to flexibly trade reconfiguration cost (time and energy) for quality to match the demands of the application. Various options are analysed, and evaluated in simulation.

1 Introduction

The research on Wireless Sensor Networks (WSNs) in recent years has focused extensively on hardware, operating systems and communication protocols. Quality-of-Service (QoS) management, in which one places constraints on performance criteria on several observable quality metrics such as lifetime, reliability, or delay, is becoming an important area of interest as well. A relatively new, but very significant topic deals with the question of how to configure a WSN that has been designed and deployed to fulfil a certain task. Sensor nodes all have a number of hardware or software settings that can be individually set. Carefully fine-tuning these parameters can yield significant performance gains. The problem is very challenging due to the vast number of possible configurations a WSN has, a number that grows exponentially with the size of the network.

In earlier work, we provided a solution to the configuration problem for static networks [1]. The solution is a scalable configuration method that intelligently searches through the full configuration space and delivers Pareto-optimal solutions or *Pareto points*: the best possible trade-off points. However, sensor networks are often dynamic, and the only way this method is able to deal with events such as moving nodes, is by completely reconfiguring the network.

This paper extends the configuration method by providing facilities for efficient reconfiguration upon a move of the data sink. Supporting a mobile sink is of interest for lifetime extension, as it relieves the energy bottleneck that naturally

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 53–68, 2009.
© Springer-Verlag Berlin Heidelberg 2009

exists at nodes close to the sink, which need to transfer much more data than nodes further away [2,3]. Furthermore, the application may have the need for a mobile sink, for example in disaster-recovery situations in which rescue workers walk around with handheld devices to collect information about the scene.

The configuration method assumes that a routing tree is used for communication between sensors and the sink. If the sink moves, the tree likely breaks, and needs to be fixed. Furthermore, the change in situation may have rendered the current configuration sub-optimal, or worse, QoS constraints may have been violated. Efficient reconfiguration of the tree and node parameters requires distributed and localised algorithms. In this paper, we describe an algorithm for tree reconstruction that is able to trade the cost of reconfiguration in terms of time and energy, for the quality of the new tree measured by the average path length. The algorithm selectively reconfigures a local area around the sink of which the size can be adjusted, while guaranteeing that a correctly rebuilt tree results. This goes hand in hand with a localised QoS-optimisation scheme that is able to find new Pareto points given that only a sub-set of the nodes may be touched. Both methods and their practical and seamlessly integrated implementation, are the main contributions of this paper.

2 Related Work

Many researchers recognise the need for methods that deal with conflicting performance demands and set up a sensor network properly. Some authors suggest to use a knowledge base to make a match between task-level demands and network protocols to use [4,5]. These efforts choose a mode of operation that is common for all nodes in the network, while our configuration method determines settings for each node individually. MONSOON [6] is a distributed scheme that uses agents to carry out application tasks, while the behaviour of these agents is adapted to the situation at hand according to evolutionary principles. Lu et al. [7] also look at WSN configuration in a method for address allocation. The overhead of the configuration protocol itself is optimised, but unlike our approach, the performance of a higher-level application is not. None of the existing configuration approaches explicitly deals with adaptation to sink movement.

Several topology-maintenance schemes have been suggested earlier [8,9], some of which aim at mobile sinks or targets [10,11]. The tree-reconstruction method of Zhang et al. [12] comes closest to our method, as they also flood a restricted area. However, our way of combining such restricted flooding with a baseline mechanism that ensures connectivity is new. By doing this, we enable a wide range of possible trade-offs between maintenance costs and task-level quality metrics. Furthermore, contrary to many existing approaches, our tree-reconstruction method does not require any knowledge about the deployment of nodes or movement of the sink, and is robust to message loss. Moreover, our way of integrating topology control with node configuration to meet task-level QoS goals is unique.

3 Configuring a WSN with QoS Requirements

In this section, we give an overview of the configuration process and the type of networks we target. Our dynamic reconfiguration method follows the same general steps. The focus of this work is on large networks of static sensor nodes positioned in some target area. In addition, there is sink node, whose job is to collect the data from the WSN. We assume all nodes have similar communication capabilities and all network links are symmetric. The network has a specific task, such as tracking a target or creating a map of the area based on measured data. We further assume that a routing tree is used to connect each node to the sink.

The process of configuring a WSN consists of a number of phases and can be executed in both a centralised or a distributed fashion. The first configuration phase is the construction of the tree. The goal of this phase is to create a tree that is good in terms of the quality of the task running on the network, but also tuned to relax the complexity of the following phase. Given the tree, the QoS-optimisation phase is responsible for finding a Pareto-optimal set of configurations (see below) for the parameters of all nodes in the network, in terms of a number of quality metrics. An algorithm to do this efficiently was introduced earlier [1]. One of the found configurations is then chosen based on constraints and other considerations, and loaded into the network (the loading phase).

3.1 Routing-Tree Construction

A routing tree has two properties that are especially relevant for the configuration problem: the average path length and the maximum node degree. Short paths in the tree are favourable for delay metrics. The degree of a node is defined as the number of child nodes it has. A low maximum node degree leads to good load balancing, and therefore to a better lifetime. Furthermore, the complexity of the QoS-optimisation algorithm greatly depends on the degree of nodes (see below). We assume that the quality of the task improves if the average path length is reduced for a given maximum node degree, and vice versa.

As minimising the average path length and maximum node degree are conflicting goals, the challenge is to find a suitable trade-off between them. We developed a method that finds a shortest-path spanning tree (SPST) within a given maximum-degree target [13]. First, the network is flooded to construct an SPST. In practise, flooding does not always result in an SPST, but here we accept this sub-optimality in exchange of efficiency and ease of implementation. Subsequently, each node that has a degree higher than the target tries to reduce its degree by requesting some of its children to find a suitable other parent node. Further details are not important for this paper, and can be found in the reference.

3.2 QoS Optimisation

Each node has *parameters*, which are hardware or software settings that can be controlled, and every parameter has a set of possible values. Suppose a node has three parameters with three possible values each; then there are $3^3 = 27$ possible

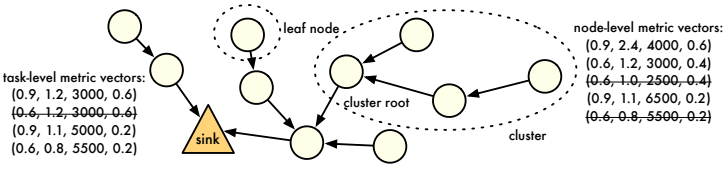


Fig. 1. Illustration of a network, cluster, leaf node, and node/task-level quality-metric vectors (tuples of information completeness, detection speed, lifetime and coverage degree); dominated configurations are crossed out

parameter vectors per node. In a network of n nodes, with $3n$ independent parameters, this leads to 27^n possible *configurations* for the whole network. Each parameter vector of a node maps to a vector of measurable properties called *quality metrics* for this node. Likewise, a parameter vector of all nodes together maps to a vector of quality metrics for the whole network, which are our optimisation targets. Such mappings can be captured in equations, and together form a model of the WSN and the task running on it. Figure 1 shows an example of a network, configurations, and related concepts.

An example of a model for a target-tracking task is available in [1]. In this task, one or more targets move around in an area of interest. Any node that detects a target, sends a report to the sink node. The model has four task-level quality metrics: *information completeness*, the fraction of the data messages from the sensor nodes that arrive at the sink; *detection speed*, a measure for the delay from detection of a target to notification at the sink; *lifetime*; and finally *coverage degree*, the minimum percentage of the time that each part of the area is observed by at least one sensor. There are inherent trade-offs between these metrics.

The challenge is to find a set of network configurations (vectors of parameters plus metrics they map to) that are *Pareto optimal*. A configuration $\bar{c}_1$ *dominates* a configuration $\bar{c}_2$ if and only if $\bar{c}_2$ is not better than $\bar{c}_1$ in any of the quality metrics. The Pareto-optimal configurations, or *Pareto points*, of a set of configurations are all configurations that are not dominated by any other configuration in the set, and are therefore considered to be the best. The process of finding all Pareto points of a configuration set is called *minimisation* [14].

An obvious way to find the Pareto points for a WSN is to compute all configurations and minimise this set. However, since the total number of configurations for a WSN typically increases exponentially with the number of nodes, this is generally not feasible. A solution is to find the Pareto points of groups of nodes called *clusters*, and incrementally grow these groups. At each step, dominated configurations are removed to keep the sets small. First, every node is initialised as a one-node cluster. This means that all parameter vectors are mapped to cluster-level metrics, and the resulting configuration set is minimised. Then, two or more clusters are combined, cluster metrics for this higher-level cluster are derived using cluster-to-cluster mappings, and the resulting configuration set is minimised. This is repeated until there is a single cluster left, which contains all nodes. The quality metrics of this cluster are the task-level quality metrics.

Algorithm 1. Distributed QoS-analysis algorithm (runs in each node)

```
1 wait until each child node has transferred its Pareto set
2 create one-node cluster Pareto set
3 combine one-node cluster and child-clusters Pareto sets (if any)
4 derive quality metrics of new configuration set
5 minimise configuration set
6 send minimised configuration set to parent
```

It is not possible to combine any arbitrary group of clusters into a new cluster, and then minimise the new configuration set, without the risk of losing configurations that are Pareto optimal at the task level. We need a property called *monotonicity*: a clustering step is monotone if and only if dominated configurations from the clusters that are being combined can never be part of a non-dominated configuration in the combined cluster. If all clustering steps are monotone, dominated configurations can be safely removed in each clustering step, without potentially losing Pareto-optimal network configurations.

It has been proven [1] that a clustering step is monotone if two conditions are met. Firstly, the cluster-to-cluster mappings need to be monotone (they need to be non-decreasing functions), which is the case for the target-tracking model. Further, every cluster must form a sub-tree of the routing tree. Therefore, a cluster is redefined to be a root node together with *all its descendants* in the tree (all downstream nodes up to and including the leaf nodes). This imposes a clustering order that starts with the leaf nodes of the network, and grows clusters towards the sink. A straightforward distributed algorithm is possible, in which each node runs the program given in Algorithm 1. After the sink completes the program, it has the complete set of Pareto points for the whole network: the quality metrics plus parameter values for each node that achieve these. Next, the sink chooses a feasible Pareto point, and sends it down the tree (the loading phase).

The run time of this algorithm for a single node is roughly proportional to the product of the number of configurations in each Pareto set that is combined, and therefore heavily depends on the number of children the node has. This is one of the reasons to build a routing tree with low node degrees. It is shown that the algorithm, while having an exponential theoretical worst-case complexity, is sub-linear in practise, and therefore scalable to large networks.

To reduce memory usage, and communication in the loading phase, an indexing method is used. Each node maintains an indexing table that links each configuration in its Pareto set to the associated configurations of its child nodes, and does not store configurations of individual nodes in its cluster. In the loading phase, a node only needs an index from its parent to know which parameters to use, and sends indices to its children after a look-up in the indexing table.

4 Adapting the Routing Tree

The previous section shows how to configure a WSN from scratch. We now consider a configured WSN in operation, in which the sink moves to a new location.

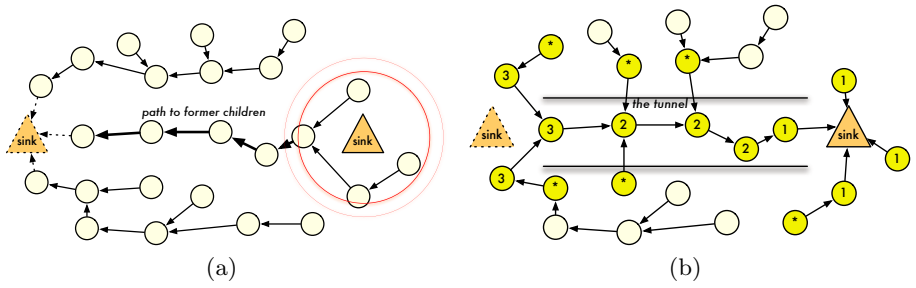


Fig. 2. The sink moved from left (dashed triangle) to right. (a) The nodes that have dashed arrows became disconnected after the move of the sink. Thick arrows indicate a path from the new position of the sink to these nodes. (b) Dark-coloured nodes form the affected area of QuickFix. The number indicates the QuickFix phase they perform.

To ensure the network's task is able to continue running, and its quality meets the demands in the new situation, a reconfiguration of the network is needed. As the routing tree most likely breaks when the sink moves, the tree needs to be reconstructed (see Figure 2(a)). We assume that the sink moves stepwise, and after each step resides at its position long enough to justify rebuilding the tree. For applications with a continuously and relatively fast moving sink, maintaining a routing tree is probably not the best solution and other methods of delivering data to the sink may be more suitable.

As outlined in Section 3.1, our goal is to create a tree in which all nodes have a degree no more than a certain threshold, and paths are made as short as possible within that constraint. After a move of the sink, the cost of globally reconstructing the tree (in time and energy) may be too high, especially if moves are frequent. We therefore propose a new algorithm that recreates the tree only in a certain region around the sink, referred to as the *affected area*, and retains the parts of the existing tree elsewhere, thereby sacrificing some quality (longer paths).

4.1 Minimal-Cost Reconstruction

We consider full tree reconfiguration as a baseline algorithm that provides the best quality against the highest cost. At the other end of the spectrum would be an algorithm that has the lowest reconfiguration cost, but a lower quality as well. This algorithm ensures that all nodes are connected to the sink again, and is therefore required for a minimum service level, but does nothing beyond that to improve the quality. We call this algorithm *QuickFix*. It is similar to the Arrow protocol introduced in a different context [15]. We first explain QuickFix, and then explain under which assumptions it works properly.

After the sink moves to a new position, it may be out of range of some or all of its children in the existing tree. QuickFix creates new paths from these nodes to the sink. All other nodes in the network are descendants of the former children of the sink. Therefore, reconnecting these former children to the sink means that all nodes are connected again. Reconnection is based on the observation that the

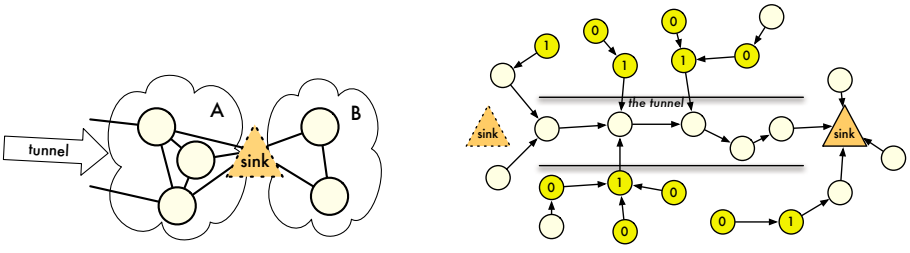


Fig. 3. (a) Former children of the sink, in two disconnected groups. Group A is connected to the tunnel, but none of these nodes can reach a node in group B by broadcasting. Group B's nodes (plus all descendants) remain disconnected after QuickFix. (b) The example of Fig. 2 with Controlled Flooding (CF) having deviation = 1. Dark-coloured nodes have been affected by CF and some of them (those in the bottom part) have found a shortcut to the sink. The numbers indicate the deviation values per node.

existing tree has paths to the sink's former child nodes from anywhere in the area, and happens in three phases; see Figure 2 for an overview. The sink, at its new position, starts by broadcasting a control message. The nodes that receive this message become the sink's new children, which all have a path to one of the disconnected former children of the sink. In the second phase, QuickFix follows these paths and reverses the links, until such a former child is reached. Finally, this node broadcasts a control message that enables other former children of the sink to connect, which is repeated until all are reconnected.

QuickFix effectively creates a *tunnel*, containing all above mentioned paths, through which all disconnected nodes are reconnected to the sink. This has no effect on the node degrees (except the sink's), but all paths are enlarged by paths of the tunnel. An optimisation that does not cost any extra transmissions can be made: any node that overhears a control message may set the sender of this message as its parent node (nodes marked by an asterisk in Figure 2(b)). By doing this, the node creates a shortcut to the tunnel and shortens the path of itself and its descendants.

QuickFix leads to a tree containing all nodes under the following conditions:

1. The sink's broadcast after the move is received by at least one sensor node. If not, the first phase breaks.
2. QuickFix messages creating the tunnel are not lost. If this happens, the second phase breaks.
3. The sub-network consisting of only the former children of the sink is fully connected. If not, the third phase breaks (see Figure 3(a)).

The first condition implies that the sink needs acknowledgements from its new children that have received its broadcast. If no acknowledgement is received, the sink rebroadcasts. The next condition can also be guaranteed by an acknowledgement scheme, as all transmissions are unicast. The third condition will generally be met if the node density (with respect to the radio range) is sufficiently high. This will usually be the case, as WSNs are typically very dense.

4.2 Improving the Quality

QuickFix reconnects all nodes to the sink in a highly cost-efficient way, but the average path length of the resulting tree will be high. The affected area consists of only the old and new children of the sink and the tunnel between them. To reduce the average path length, but keep the costs limited, we use another mechanism on top of QuickFix, which enlarges the affected area by a number of hops that can be specified, called the *deviation* parameter. This parameter is part of the control-message format. Any node that overhears a control message does not only connect to the sender (as described in the previous sub-section), but if the deviation value is larger than zero, it will broadcast a new control message with a decremented deviation value. We refer to this as *Controlled Flooding* (CF). By flooding the affected area, a local SPST is constructed, and consequently also the paths of the nodes outside the area are reduced in length (see Figure 3(b)).

QuickFix and CF are not executed consecutively, but run in parallel. To ensure that nodes react to a control message only once, an update number is used in the control-message format. This number is incremented at each reconfiguration (sink move), and only if a node receives a control message which has a higher update number than it has seen before, it will update its parent variable and forward the message. There is one exception to this rule: since QuickFix is crucial to reconnect all nodes in the new tree, QuickFix control messages are always forwarded (to the parent in the *old* tree!), even though a CF message with the same update number arrived earlier. Since all chains of forwarded control messages (QuickFix and CF) originate from the sink, each affected link is pointed to the node that sent the message, and nodes react only once to CF messages of a certain update number, a correct tree (rooted at the sink, loop-free) is formed in the affected area. The nodes at the edge of the affected area keep their existing sub-trees, so all nodes are connected to the affected area and hence to the sink. Loss of CF messages may lead to longer paths, but never results in a broken tree.

After QuickFix and CF finish, the node degrees are reduced as before. Only nodes in the affected area take part in the degree reduction. When only QuickFix is used, the reduction algorithm is not able to do much, since the affected area is small. Therefore, we bound the number of nodes that may directly connect to the sink already in the first phase of QuickFix via some extra handshaking.

The deviation parameter controls the trade-off between reconfiguration cost and quality. A larger deviation value leads to a larger affected area, and thus to more nodes obtaining shorter paths, and a better quality. On the other hand, reconfiguring a larger affected area takes more time and more transmitted control messages. The best value for the deviation parameter depends on the application.

5 Reconfiguring Node Parameters

Normal operation of the network task can continue as soon as the tree has been reconstructed. However, due to the changes in the structure of the network, the level of quality achieved by the running task is typically lower than possible, and

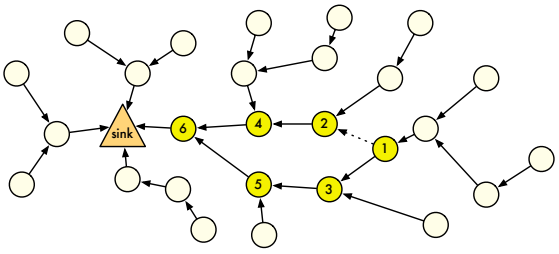


Fig. 4. Node 1 has changed from node 2 to node 3 as its parent. Only the dark-coloured nodes and the sink (the *affected area*) need to recompute their Pareto points.

could even be such that QoS constraints are violated. It is therefore worthwhile to improve the quality by reconfiguring the nodes' parameters.

While parameter reconfiguration is in progress, the network is in a state of reduced quality. It is therefore desirable to reconfigure as quickly as possible. Moreover, parameter reconfiguration comes at a cost, as processing and communication is needed to compute and load the new settings. In this section, we explore the trade-off between the quality achieved by reconfiguration and the cost it has.

5.1 Optimisation Strategies

We first discuss how to find the best possible node configurations in terms of quality, given the reconfigured tree. The most straightforward, but inefficient, way to do this is to simply re-run Algorithm 1 on all nodes, from the leaves up to the root. Observe however, that if the tree reconfiguration only happens in a local area around the sink, many nodes and their sub-trees/clusters remain unchanged. Therefore, also the sets of Pareto-optimal configurations for all nodes and clusters outside the affected area do not change, and need not be recomputed.

To verify this, first consider a fully configured network in which a single node changes its parent (for whichever reason), as in Figure 4, where node 1 switches from node 2 to 3. This would cause changes in the Pareto set of the cluster with root node 1. Further, the roots of all other clusters that have changes in them need to be updated (remember that a cluster is a node with *all* its descendants): the clusters with as root node 1, its old and new parent (nodes 2 and 3), and all nodes on the paths from these three nodes to the sink (nodes 4, 5 and 6). This implies that the QoS-optimisation algorithm may start at the nodes at the edge of the affected area (further referred to as the *boundary nodes*; in Figure 4, nodes 1 and 2 are boundary nodes) instead of at the leaf nodes of the network. The reconfiguration of the affected area reuses the Pareto sets of the clusters just outside the area. Note, however, that the newly selected configuration at the sink, may cause a different configuration to be selected from the Pareto sets of the nodes outside the affected area. This means that, while recomputing the Pareto sets is local, loading the selected configuration still involves all nodes in the network.

To make the reconfiguration completely local, not only the tree reconstruction and QoS analysis phases, also the loading phase should be restricted to a local

area. Nodes outside the affected area should retain their configurations, and this should be taken into account in the QoS-analysis: not the full Pareto sets of the non-changing clusters should be used, but only the selected configuration. A boundary node then combines its new one-node cluster set with the selected configurations of each of its children. This has the added benefit that the analysis becomes simpler (smaller configuration space), significantly reducing the cost of reconfiguration. The price is sub-optimality of the found task-level Pareto set and hence a potentially non-optimal quality of the selected configuration. To further exchange quality for lower cost, we could reduce the area even more (smaller than the area of tree reconfiguration), the extreme case being not reconfiguring at all. However, not re-analysing all nodes in the affected area means that the computed task-level metrics are not accurate; when locally reconfiguring from boundary nodes to root, the computed metrics are always accurate.

5.2 Practical Details

From the previous, it follows that we need to make the boundary nodes start Algorithm 1 with the correct child Pareto sets. However, a node actually does not know whether it is a boundary node or not, and what is more, not every node knows that it is a part of the affected area. In the example of Figure 4 in which node 1 changed parents, only nodes 2 and 3 will be aware of the change, while also 4, 5, and 6 need to be updated. We therefore make every changed node send a message to its parent (after some delay to ensure the tree is stable) that indicates it is part of the affected area. If the parent was not a changed node, it now knows that it is also in the affected area, and forwards the message to its own parent. Subsequently, to start the analysis process, a node outside the affected area that overhears such a message from its parent (all light-coloured nodes in Figure 4 that are children of dark coloured-nodes), knows that its parent is a boundary node, and forwards its unchanged cluster-level Pareto set, or only the currently selected configuration in case of localised reconfiguration. Nodes in the affected area can now continue as in Algorithm 1.

After completing the QoS-analysis phase, the sink proceeds with the loading phase as usual. When the load messages reach outside the affected area, they are no longer forwarded in the localised case. In the globally-optimal case, the load messages are forwarded until the leaf nodes of the network.

6 Experiments

Since the performance of our reconfiguration approach depends on many factors, such as the type of task, the size of the network, and the movement pattern of the sink (size of steps, speed), we use simulations to compare various scenarios. We are especially interested in the influence of the deviation parameter and the choice between localised and globalised QoS analysis on resulting task quality and reconfiguration costs. To see whether parameter reconfiguration really makes sense, we also compare the results with the option of not reconfiguring at all (but we do always need QuickFix, as the tree always needs to be fixed after a move).

6.1 Simulation Overview

All simulations were done in the OMNeT++ discrete-event simulator [16], for networks of 900 TelosB sensor nodes [17] randomly deployed in an area of 300×300 m. These nodes employ a simple 8 MHz processor and a radio transceiver with 250 kbps bit rate. To ensure an even distribution of the nodes in the area, they were placed with some variance around fixed grid points. The communication range of the nodes was set to 20 m. The first scenario we look at has the sink placed at coordinates (100,150), and the network configured with the existing method [1]. Then the sink makes a single move to position (200,150), and the network is reconfigured using the various options described in this paper. We simulated 100 different networks and report the medians of the metrics of interest. In the second scenario, the sink makes multiple consecutive moves, while the network is reconfigured after each move.

Our simulations take into account the processing time of Algorithm 1 on real TelosB sensor nodes. Profiling of a TinyOS implementation on such a node was done to obtain the run times for various sizes of the configuration sets (see [13] for details). All simulations were done for the target-tracking task [1] and 27 different configurations per node. We distinguish four cases: naive and efficient global reconfiguration, local reconfiguration, and no reconfiguration at all. The former two refer to re-analysing all nodes and only the affected nodes respectively, which should both arrive at the same globally-optimal Pareto points.

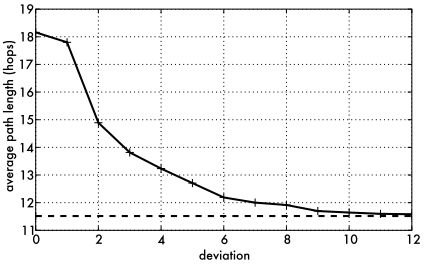
A selection function is needed that chooses one of the Pareto points to be loaded into the network. For easy comparison of the various methods, we use a selection function that assigns a single value to a configuration. We use a weighted sum of all four metrics, where each weight normalises the metric. To this end, we define the *value* of a configuration vector $\bar{c}$ = (information completeness, detection speed, lifetime, coverage degree) (see Section 3) as its inner product with the vector $\bar{v} = (100, 200, 0.1, 100)$.

The evaluation metrics are as follows:

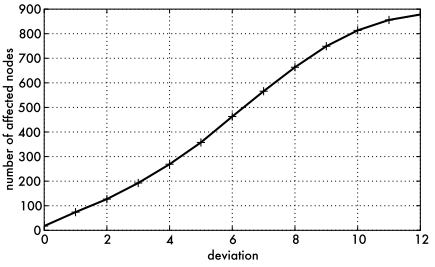
- **Disruption time**: the duration of service disruption just after the sink's move until the tree has been reconfigured, and thus equal to the time needed for tree reconstruction. During the parameter-optimisation time, the network does function, though its service quality is reduced.
- **Reconfiguration time**: the total duration of the tree- and parameter-reconfiguration process. The total reconfiguration time is also a rough indication of the amount of processing needed on the nodes (the optimisation time is dominated by processing).
- **Communication cost**: the average number of bytes transmitted for reconfiguration, over all nodes in the network.
- **Value loss**: the relative loss in value compared to the best case.

6.2 Tree Reconstruction

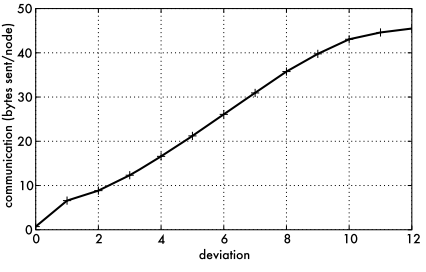
We first study the behaviour of the tree-reconstruction algorithm described in Section 4 in the single-move scenario. All 100 networks were tested with various



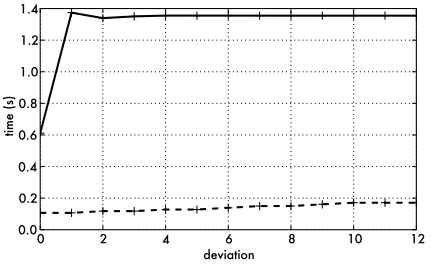
(a) Average path length (the dashed line is the optimum)



(b) Size of the affected area



(c) Communication cost

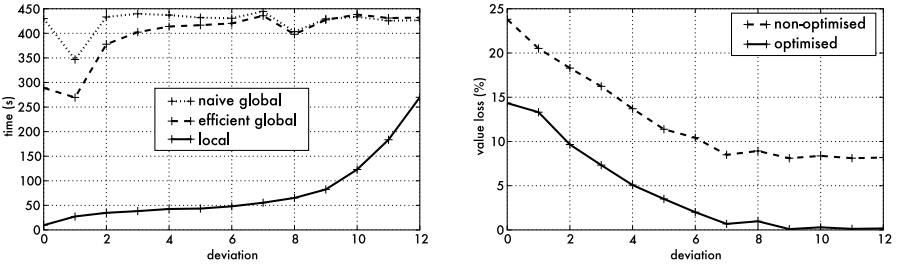


(d) Total disruption time (solid line) and only QuickFix+CF (dashed line)

Fig. 5. Evaluation of tree reconstruction for various deviation values

values of the deviation parameter, as well as full flooding as a baseline. The degree-reduction algorithm with a target node degree of 2 (see Section 3.1) was executed on the resulting networks. The first point to note is that the tree was correctly rebuilt in all of the cases. Figure 5(a) shows that average path length decreases monotonically from almost 18.2 to 11.5 when increasing the deviation from 0 (only QuickFix) to 12. The optimal average path length (when fully flooding the network) is also 11.5. Figure 5(b) indicates that the size of the affected region also grows steadily with the deviation until, at deviation 12, almost the whole network is reconfigured, and hence we obtain an SPST with this deviation (within the degree constraint). Observe that the affected area first grows faster than linearly, but slows down after deviation 6; this is the point where Controlled Flooding reaches the edges of the network.

Along with the affected area, the amount of communication increases in a similar pace (Figure 5(c)). As expected, also the time it takes to rebuild the tree, excluding degree reduction, increases with the deviation (Figure 5(d), dashed line), with an offset due to QuickFix. The time needed for QuickFix/CF is relatively short compared to the time used to reduce the node degrees, and together with the fact that the latter does not depend on the size of the affected area, this explains why the total time spent on tree reconstruction (solid line in Figure 5(d)) is not clearly dependent on the deviation. Also recall that if only



(a) Optimisation time (b) Value loss

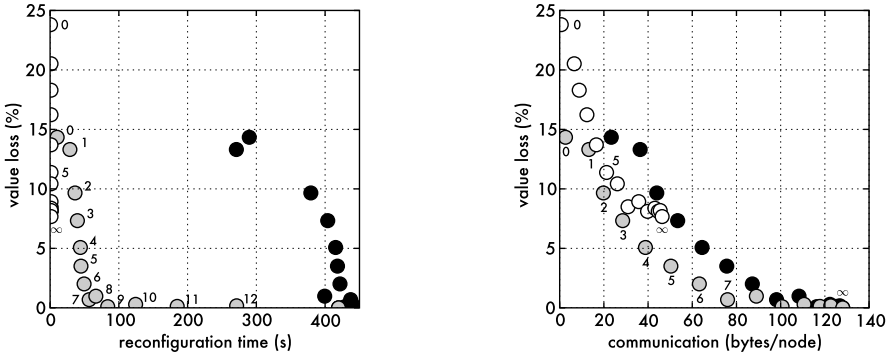
Fig. 6. Timing results and value improvement for parameter optimisation

QuickFix is used, the degree-reduction algorithm is hardly effective due to the small affected area, which is why the run time at deviation 0 is lower than for the others. Overall, the total disruption time is always less than 1.4 s.

6.3 Parameter Optimisation

Now compare the optimisation times of the various cases of parameter optimisation in Figure 6(a). We confirm that efficient global reconfiguration, in which only nodes in the affected area recompute their Pareto points, is always faster than the naive version, and this is most pronounced for small deviations. However, the differences are not as large as might be expected. Local optimisation on the other hand, in which the same nodes recompute their Pareto points as in the efficient global case, but with boundary nodes using just one configuration for their child nodes outside the affected area (instead of their full set of Pareto points), is much faster. This may imply that the configuration sets of nodes closer to the sink are larger than those of nodes further away. It is interesting to see that the optimisation time of the localised algorithm initially increases very slowly (sub-linearly) with the deviation, starting at just 9.7 s. Deviation 5 appears to be an inflection point beyond which the rate of increase grows quickly. Eventually, the three lines meet at 418 s (about seven minutes; not visible in the graph), when fully flooding the network; this is equivalent to deviation infinity, as the affected area is the whole network.

Next, we test the quality of the resulting configurations by comparing their values. The best value occurs when using full flooding and global parameter optimisation. Using this value as a baseline, Figure 6(b) shows the relative loss in value when using the other methods. It turns out that, for the target-tracking task, all methods for parameter optimisation, local and global, achieve the same quality (the solid line), while not optimising is significantly worse (8 to 10 percentage points; the dashed line). Given its low overhead, this makes local reconfiguration very attractive for any deviation. The best value possible when not optimising (at deviation 12) can be attained with optimisation already at deviation 3. For a larger deviation, we see a steady improvement in value, which is consistent with our assumptions in Section 3. At deviation 0, the difference with the optimum is quite large at 14.3%, but after deviation 6 it becomes smaller than 1%.



(a) Time vs. value loss (b) Communication vs. value loss

Fig. 7. Pareto plots for the reconfiguration process. Black dots belong to (efficient) global parameter reconfiguration, grey to local, and white to no parameter optimisation. Deviation values are given at interesting trade-off points.

6.4 Quality/Cost Trade-Offs

Combining all results yields the totals for the reconfiguration process in the evaluation metrics. The disruption time only depends on tree reconstruction and has been reported above. Figure 7 gives an overview of the trade-offs between the total time and communication costs of reconfiguration, and the value loss of the resulting configuration, for all reconfiguration options. The two plots should be seen together as a three-dimensional trade-off space. It is immediately clear that all global-reconfiguration points (the black dots) are dominated by the locally optimised (grey) and non-optimised (white) options. In contrast, most of the other points are Pareto optimal. As non-optimisation is obviously the fastest, it is the best choice when speed and low processing costs are most important, although the loss in value is at least 7.7%. In many cases, however, local reconfiguration provides the best trade-off between the three metrics: low cost and good quality. At deviation 5, for example, local reconfiguration takes 44.9 s (of which the service is disrupted for 1.4 s), costs 50.3 bytes of communication per node, and the overall quality is 3.5% lower than the best case. The configuration space that was analysed in this time, for the affected area of 350 nodes, has a size of 27^{350} configurations, of which the found configuration has the optimal value.

6.5 Multiple Moves

It is interesting to see what happens to the loss of value when the sink moves repeatedly, and the network is locally optimised at each move. Figure 8 shows the value loss of local optimisation with deviation 5 compared to the optimal case, for 25 consecutive moves of 50 m in random directions. The results are averages over five different runs. A very irregular pattern is visible, but the trend is a slowly increasing loss for each move. We therefore suggest to do a full network reconfiguration periodically, or when the attained value becomes too low.

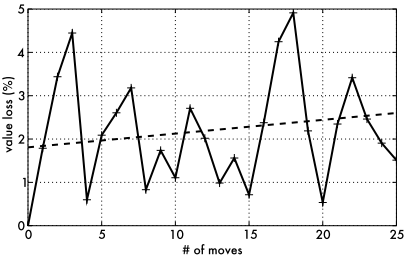


Fig. 8. Multiple consecutive moves: value loss compared to optimal with trend line

7 Conclusions and Future Work

This paper provides a method for reconfiguration of a WSN with a mobile sink. It does not only reconstruct the routing tree, but also optimises the parameters (hard- or software settings) of the nodes that were affected to improve the service level. Since reconfiguration takes time and energy, which are scarce resources, optimally reconfiguring the network each time the sink moves is likely to be infeasible. Trade-offs can be made between the effort spent on optimisation and the resulting level of service quality of the running task in terms of QoS metrics. The algorithms work in a distributed and localised way by reconfiguring only an area around the sink, of which the size can be adjusted via a deviation parameter. Practical implementation details are given, and experiments show that the localised algorithms indeed manage to find suitable trade-off points.

The best choice of reconfiguration method and deviation value heavily depends on the application and its requirements, and the sink behaviour. Due to the unpredictable nature of the sizes of the Pareto sets and therefore the optimisation time, as well as the quality of the resulting task-level Pareto set, it is hard to give guidelines for this choice. In practise, a system could first have a calibration phase to tune the deviation value, or simulations like ours can be used.

The methods that do all processing in-network are useful for applications in which the sink stays at its position for a while before moving again (e.g. when moving the sink for lifetime improvement), to justify the cost and speed of reconfiguration. For scenarios such as disaster recovery, in which the sink (rescue worker with handheld) may move a bit faster, an interesting option is to deploy special, more powerful nodes that handle most of the optimisation duties, or even do all the work at the sink. This is again a trade-off: between communication and processing cost (offloading computation effort increases the amount of communication between sensors and sink), and of course the cost of the additional nodes. For example, doing the QoS analysis for deviation 5 as above on a laptop (Intel Core 2 Duo processor at 2.4 GHz) takes 3.0 s for the globally-optimal case, and just 0.6 s for the localised case. For handheld devices, these numbers would be higher, but still much lower than when done in-network on sensor nodes. However, the communication costs per node increase by about five times (both global and local). Future work will focus on this trade-off in more depth.

Acknowledgements. The authors would like to thank Dr. Vikram Srinivasan for his help in the early stages of this project. This work was partly supported by EC FP6 project IST-034963.

References

1. Hoes, R., Basten, T., Tham, C.K., Geilen, M., Corporaal, H.: Quality-of-service trade-off analysis for wireless sensor networks. Elsevier Performance Evaluation (2008), http://dx.doi.org/10.1016/j.peva.2008.10.007
2. Luo, J., Hubaux, J.P.: Joint mobility and routing for lifetime elongation in wireless sensor networks. In: INFOCOM 2005, Proc. IEEE, Los Alamitos (2005)
3. Wang, W., Srinivasan, V., Chua, K.C.: Using mobile relays to prolong the lifetime of wireless sensor networks. In: MobiCom 2005, Proc., pp. 270–283. ACM, New York (2005)
4. Pirmez, L., Delicato, F., Pires, P., Mostardinha, A., de Rezende, N.: Applying fuzzy logic for decision-making on wireless sensor networks. In: Fuzzy Systems Conference 2007, Proc., pp. 1–6. IEEE, Los Alamitos (2007)
5. Wolenetz, M., Kumar, R., Shin, J., Ramachandran, U.: A simulation-based study of wireless sensor network middleware. Network Management 15(4), 255–267 (2005)
6. Boonma, P., Suzuki, J.: MONSOON: A coevolutionary multiobjective adaptation framework for dynamic wireless sensor networks. In: HICSS 2008, Proc., pp. 497–497. IEEE, Los Alamitos (2008)
7. Lu, J., Valois, F., Barthel, D., Dohler, M.: Fisco: A fully integrated scheme of self-configuration and self-organization for wsn. In: WCNC 2007, Proc., pp. 3370–3375. IEEE, Los Alamitos (2007)
8. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In: INFOCOM 2002, Proc., pp. 23–27. IEEE, Los Alamitos (2002)
9. Schurgers, C., Tsiatsis, V., Srivastava, M.B.: STEM: Topology Management for Energy Efficient Sensor Networks. In: Aerospace Conference, Proc. IEEE, Los Alamitos (2002)
10. Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Hubaux, J.-P.: MobiRoute: Routing towards a mobile sink for improving lifetime in sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 480–497. Springer, Heidelberg (2006)
11. Bhattacharya, S., Xing, G., Lu, C., Roman, G.C., Harris, B., Chipara, O.: Dynamic Wake-up and Topology Maintenance Protocols with Spatiotemporal Guarantees. In: IPSN 2005, Proc. IEEE, Los Alamitos (2005)
12. Zhang, W., Cao, G.: Optimizing tree reconfiguration for mobile target tracking in sensor networks. In: INFOCOM 2004, Proc. IEEE, Los Alamitos (2004)
13. Hoes, R.: Configuring Heterogeneous Wireless Sensor Networks Under Quality-of-Service Constraints. PhD thesis, TU/e and NUS (to appear, 2009)
14. Geilen, M., Basten, T., Theelen, B., Otten, R.: An algebra of Pareto points. Fundamenta Informaticae 78(1), 35–74 (2007)
15. Demmer, M., Herlihy, M.: The arrow distributed directory protocol. In: Kutten, S. (ed.) DISC 1998. LNCS, vol. 1499, pp. 119–133. Springer, Heidelberg (1998)
16. OMNeT++, http://www.omnetpp.org
17. Crossbow Technology, http://www.xbow.com

A Context and Content-Based Routing Protocol for Mobile Sensor Networks*

Gianpaolo Cugola and Matteo Migliavacca

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{cugola,migliava}@elet.polimi.it

Abstract. The need of monitoring people, animals, and things in general, brings to consider *mobile WSNs* besides traditional, fixed ones. Moreover, several advanced scenarios, like those including actuators, involve *multiple sinks*. Mobility and multiple sinks radically changes the way routing is performed, while the peculiarities of WSNs make it difficult to reuse protocols designed for other types of mobile networks.

In this paper, we describe CCBR, a *Context and Content-Based Routing* protocol explicitly designed for multi-sink, mobile WSNs. CCBR adopts content-based addressing to effectively support the data-centric communication paradigm usually adopted by WSN applications. It also takes into account the characteristics (i.e., context) of the sensors to filter data.

Simulations show that CCBR outperforms alternative approaches in the multi-sink, mobile scenarios it was designed for, while providing good performance in more traditional (fixed) scenarios.

1 Introduction

The recent advances in WSNs are rapidly expanding the range of applications they can support: from "traditional" environmental monitoring, where a number of nodes is scattered across an area collecting data for a single sink, to *mobile scenarios* involving *multiple sinks*. This happens when the entities to monitor are animals (e.g., in farming scenarios), people (e.g., in elderly care scenarios), or things moving around (e.g., in logistics), while several mobile devices (e.g., PDAs) are used as sinks, or actuators, also acting as sinks, are involved.

Unfortunately, mobility and the presence of multiple sinks is something that has been largely neglected by research on WSNs so far, especially if we consider the case of data-aware routing protocols. Indeed, one of the main peculiarities of WSNs is the data-centric form of communication that they usually adopt: a few *sinks* (a single one in the simplest scenarios) are interested in receiving only some specific data among those collected by sensors, e.g., the temperature readings that exceed some threshold. This suggests abandoning traditional, address-based routing protocols, to adopt a *Content-Based Routing* (CBR) [1] protocol, in

* This work has been partially funded by the European Community under the IST-034963 WASP project and by the Italian Government under the MIUR-FIRB program INSYEME.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 69–85, 2009.
© Springer-Verlag Berlin Heidelberg 2009

which messages do not carry any explicit address, while they are routed based on their content and on the interests specified by nodes. This is the solution taken by popular protocols for WSNs like Directed Diffusion [2] and TinyCOPS [3], without however considering mobility as a key aspect.

Moreover, if we look at the typical scenarios of usage of a WSN, we may notice that communication is not only data-centric but it is also often *context-aware*. As an example, a farmer could be interested in knowing the activity level of "young" cattle only, while, in a logistics application, different temperature thresholds are critical for different goods. Encoding such context-awareness as part of the message content and using standard CBR to route messages is possible, but can be inefficient, increasing message size and communication overhead.

Starting from these considerations, we developed *CCBR*, a *Context and Content-Based Routing* protocol for *multi-sink, mobile WSNs*. It adopts a probabilistic, receiver-based approach to routing, where each node decides autonomous-ly if forwarding packets, loosely collaborating with others in keeping routing information up-to-date. This approach has proven to be well suited to support the multi-sink, mobile scenarios we target, also being able to efficiently address the (albeit slower) dynamics inherent in traditional, non-mobile WSNs.

The next section goes into the details of the protocol, explaining how it operates, while Sect. 3 evaluates the performance of CCBR through simulation, comparing it with other approaches. Finally, Sect. 4 surveys related work and Sect. 5 draws some conclusions and describes our future plans.

2 Context and Content-Based Routing

The reference scenario for CCBR is that of a WSN composed of a set of *nodes* possibly moving around at different speeds. Among these nodes we distinguish between those that are interested in receiving messages (the *sinks*), and those that send out messages (the *sensors*). In such a network: (i) the same node may act both as a sink and as a sensor, and (ii) messages do not necessarily carry sensors readings. As an example, a node could collect temperature readings from other nodes (acting as a "sink") to notify fire alarms (acting as a "sensor").

2.1 The API

CCBR offers three main primitives to the upper layers:

```
setComponentProp(Properties p, DataListener dl)
listenFor(CompFilter cf, MsgFilter mf, AddData ad, MsgListener ml, int l)
send(Message m)
```

where the `DataListener dl` and the `MsgListener ml` are pointers to callback functions invoked when additional data and messages arrive, respectively.

The `setComponentProp` operation allows sensors to specify the *properties* that describe the context in which they are, e.g., the fact that they are installed on a pig or a cattle, and the age of the animal.

The `listenFor` operation allows sinks to express their interests, by specifying both the content of the messages they are interested in (through the *message filter* `mf`) and the sources they consider relevant (through the *component filter* `cf`). As an example, a sink could be interested in receiving messages such that: `activity!="stationary"` (the message filter), originating from sensors such that: `type==cattle AND age<24` (the component filter). The *additional data* `ad` is blindly transported by the protocol from the listening sinks to the matching sensors (those whose properties match the component filter). As an example, this data could be used by a sink to spread around information about the sampling period for sensing. Finally, the integer `l` is the *lease time* after which the expression of interest expires.

The `send` operation allows messages to be sent to the interested sinks, if any.

2.2 The Protocol in General

To support the mobile, multi-sink scenarios it has been conceived for, CCBR abandons the traditional approach to routing, which uses link-layer unicast packets to transport data from hop to hop, to use the broadcast facility provided by wireless networks. It also turns away from the usual, deterministic, sender-based approach to routing, to adopt a probabilistic mechanism to decide if and how packets are forwarded, leaving this decision to the receiver of the packet, which operates autonomously w.r.t. the sender.

All these choices, which differentiate CCBR from the previously proposed routing protocols for WSNs, were strongly influenced by our experience with mobile ad-hoc networks [4,5,6,7], which convinced us that "broadcast", "probabilistic", and "receiver-based" are the right keywords when mobility and multicast interactions (resulting from the presence of multiple sinks) enter the picture.

To describe CCBR in details we first describe its forwarding mechanism, i.e., how packets flow from sensors to sinks using the various routing tables, then we describe how such tables are built and maintained.

2.3 Forwarding

In CCBR, each sink has an associated *sink number*: an integer in the range $1...K$ where K is the maximum number of allowed sinks (see Sect. 2.4 for details on choosing sink numbers). To forward data from sources to sinks, each node maintains a *distance table* and a *content table*. The former stores an estimate of the distance (in hops) of the node from each sink, while the latter keeps track of the interests of sinks that are relevant for the node. In particular, the content table of a node with properties p maps each sink number n with those (not yet expired) message filters issued by n whose component filter matched p.

Figure 1 shows the content table of a sensor N with properties p when two sinks S_1 and S_2 (with sink numbers n_1 and n_2, respectively) have invoked the `listenFor` primitive, with message filters mf_{S1}, mf_{S2} and component filters cf_{S1}, cf_{S2}, both matching p. Note that N's content table does not include any information about sinks, like S_3 in the figure, whose interests do not match N's

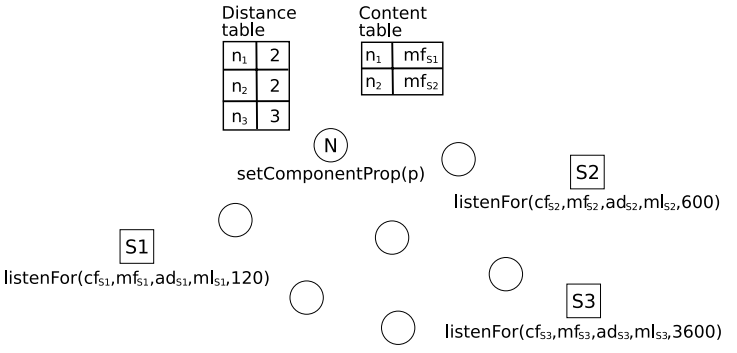


Fig. 1. Content and distance tables in CCBR

properties (i.e., cf_{S3} does not match p). This highlights the positive consequence of keeping context and content information separate: the routing tables can be kept smaller and they have to include the message filters but not the context filters. This saves memory and reduces the matching effort (and related power consumption) at message sending time, two fundamental issues in WSNs.

Whenever the `send(m)` primitive is invoked at a node N, the CCBR protocol looks up m's content in the content table and computes the set of sinks interested in receiving m. After that, it builds a *forwarding header* composed of a unique message identifier, a *destination vector*, and a *distance vector*.

- The destination vector is a bit-vector with length K, having a 1 in each position that corresponds to the number of an interested sink (as computed from the content table). In our example, supposing m matches mf_{S1} but not mf_{S2}, the destination vector has bit n_1 set, the others clear.
- The distance vector is an array of bytes, one for each interested sink, storing the distance of N from that sink (in order of sink numbers). In our example, it includes a single byte with an initial value of 2.

Afterward, CCBR builds a *forwarding packet* putting together the forwarding header and the message m and yields it to the MAC for broadcast transmission.

The first time a node receives a forwarding packet p, it compares the destination and distance vectors in the header of p with its own distance table. If the receiving node is farther or equally distant from the recipients of p, it drops it. If the receiving node is closer to at least one of the sinks listed in p's forwarding header: (i) it updates the distance vector in p, putting its own distance for those sinks it is closer, and (ii) it schedules the packet for transmission. The packet, indeed, is not transmitted immediately, while a *delay and cancel* mechanism is exploited[1].

Delay and Cancel. The packet is put in a *transmission queue*, where it remains for a period d_{tx} (the *delay of transmission*), which is smaller when the global

[1] To maximize the network lifetime we also add a probability of re-forwarding based on the remaining capacity of the node's battery.

improvement performed on the packet's distance vector is higher. As an example, consider the case of a node N with distance table $\{1|4,\ 2|2,\ 3|5\}$. When N receives a packet with destination vector 111 and distance vector $\{5,\ 3,\ 4\}$, it rewrites the latter putting $\{4,\ 2,\ 4\}$, and calculates a global improvement $H = 2$. At this point N will schedule the packet for retransmission with a delay $d_{tx} = \delta \cdot [\max(0, H_{max} - H) + rnd(0, 1)]$, where δ is proportional to the average time to transmit a packet (including MAC and transmission delays), while H_{max} has to be chosen looking at the average number of destinations for each packet. Whenever a node receives a packet, it looks at its transmission queue and deletes pending retransmission of the same packet (same identifier) if the queued copy has a distance vector which is higher or equal to the distance vector of the received packet for all destinations.

Under ideal conditions, this results in an efficient, greedy forwarding algorithm, which: (i) suppresses redundant transmissions (i.e., those originating from nodes equally distant from all the destinations); and (ii) favors, as forwarders, the nodes with the lowest distance from the highest number of destinations (i.e., the paths that lead to multiple destinations). In real scenarios, these results are only partially achievable since not all the receivers of a packet may hear each other, which may result in multiple path forwarding. We will come back on this issue while evaluating CCBR's overhead.

Mobility and Local Minima. Mobility is another factor that may break the mechanism above, by producing local minima in the distance function. This occurs whenever a node N has an wrong estimate of its distance from a sink, e.g., because it was once closer to it but now moved in a region where the real distance is higher. Nearby nodes will not forward packets generated by N because of the (wrong) lower distance it puts in the distance vector of those packets.

To solve this issue we complemented the basic forwarding algorithm above with a *retransmission mechanism*. After transmitting a packet (either as a source or as a forwarder), a node N puts it in an *retransmission queue*. If a predefined *timeout of retransmission* expires without hearing the same packet with at least one element in the distance vector lowered (i.e., if no one re-forwards the packet toward a destination), the node N:

1. adds a *retransmission bit-vector* to the forwarding header of the packet, with a one for each sink whose distance was set by itself;
2. increases the distance vector of the packet for each sink in the retransmission bit-vector;
3. transmits the resulting packet again.

Nodes hearing such a packet will reconsider it even if they already received it before, but only for improvement with respect to the sinks listed in the retransmission bit-vector (which is reset to 0, afterward).

The consequence of this mechanism is twofold: on one hand it increases the CCBR's resistance to collisions, which is good since link-layer broadcasting is particularly subject to collisions. On the other hand, increasing the distance vector for packets that where not forwarded by neighbors (step 2 above) also

allows to overcome local minima, by increasing the set of potential forwarders for the retransmitted packet.

Unfortunately, asymmetric links may trigger this retransmission mechanism even when it was not required, thus increasing the network traffic without any positive effect on delivery. To limit this problem we allow each node to retransmit each packet at most once. Moreover, we also introduce in CCBR a mechanism of *credits*, which further reduces retransmission. When a packet is created, it is assigned a predefined credit: an integer stored into its forwarding header, which is decremented each time the retransmission timeout expires at a node. The retransmission mechanism does not apply to packets which ended their credit, i.e., the initial credit of a packet represents the maximum number of times the retransmission mechanism may fire along its route from the sender to the sink.

Delivery to Sinks. Whenever a sink receives a packet p, it looks at the bit in p's destination vector that corresponds to its sink number. If the bit is set the packet is forwarded to the application layer (by invoking the corresponding `MsgListener`) otherwise it is not. In any case, p is also processed normally for forwarding (sinks are standard nodes, so they must participate in the forwarding process). Finally, if p was targeted to that sink and it is not scheduled for forwarding, a special packet is created and broadcasted to stop the retransmission mechanism at the sender node.

2.4 Routing

To build and maintain distance tables, each sink periodically (every t_b seconds) broadcasts a beacon that contains its sink number, a sequence number and a distance, initially set to 0. Each node receiving a beacon, first increments the included distance and uses it to update its distance table, then it schedules the beacon for forwarding. Even in this case we use a "delay and cancel" mechanism to limit redundant transmissions. This time we are interested in favoring beacon retransmission by nodes that are farther away from the previous forwarder, to cover more distance with fewer retransmissions. Accordingly, we use a delay that is inversely proportional to the RSSI information provided by the MAC.

Whenever the `setComponentProp` primitive is invoked at a sensor, the CCBR layer simply stores the component property and the `DataListener` internally.

When the `listenFor` is invoked at a sink, its parameters (component filter, message filter, additional data, and lease time), are stored in a *filter table*. Every t_f ($t_f \geq t_b$) seconds the filters and additional data in such table for which the lease time is not elapsed are grouped and piggybacked on top of the next emitted beacon. When a sensor N receives this special "fat" beacon, it updates its distance table but also its content table as follow: for each filter whose component filter matches N's properties, the message filter, together with the sink number, are stored into the content table. If they were not already there (i.e., the same information has not been already received before), the additional data is passed to the related `DataListener`. Moreover, old filters from the same sink that are

not refreshed by the beacon are deleted from the content table as this means that they expired.

Finally, to decide its sink number, at startup time each sink waits at least t_b seconds (but possibly a multiple of that) to see the beacons coming from other sinks. Afterward, it randomly picks a number in the interval $1...K$ (see beginning of Sect. 2.3), not chosen by other sinks and starts operating. Clashes in choosing sink numbers may still happen (e.g., if two sinks turn on at the same time). They can be easily resolved by letting the sink with the lowest MAC address (or any other distinguishing value) to change its number when it discovers the clash.

3 Evaluation

Due to the difficulty of extensively testing a protocol like CCBR in a real setting, with hundreds of nodes moving around, and to do so in a replicable way, we decided to use a network simulator. Accordingly, we implemented the entire CCBR protocol (and a model of the CC2420 card and 802.15.4 MAC) in OMNeT++ [8], using the Mobility Framework [9] to simulate a mobile environment. We used a path loss channel model fully considering interferences from other, parallel transmissions to calculate (at run-time) the SNR of each frame.

Besides measuring the performance of CCBR under different conditions, we were also interested in comparing it with other protocols. In particular, we chose two simplified protocols, which well represent two very different classes of solutions to route packets in a mobile network. We call them *Gossip* and *Uni*.

Gossip is a structure-less protocol in which nodes send packets using the broadcast facility provided by the MAC layer and forwards them based on a pure probabilistic decision: when a node hears a packet for the first time it retransmit it with a probability $p \in (0, 1]$. Gossip is interesting as it represents the simplest possible approach to manage a mobile network, indeed it is often used as a baseline to compare protocols for MANETs.

Uni adopts a totally different approach. It uses beacons flooding the network as in CCBR to build unicast routing tables toward the sinks. Messages matching the interests of a sink S (considering both its context and content part) are sent, using link-layer unicast transmissions, hop by hop, up reaching S (different sinks are managed separately). Unicast is a good representative of those protocols, like Directed Diffusion [2], which set up a tree along which sources report their data to sinks.

To evaluate the performance of CCBR under different conditions, we considered a wide range of scenarios by changing the different parameters of our simulation: the density of the network (number of sensors per Km^2), the number of sinks (one of them stays firm at the center of the field, the others move around), the area of the field in which nodes move, the pattern of mobility (including the speed at which they move), the frequency at which each sensor sends out messages, and the selectivity of filters. In all the resulting scenarios we measured the performance of CCBR, Gossip, and Uni varying their key parameters: for CCBR the beaconing interval (the filters summarizing the interests of sinks

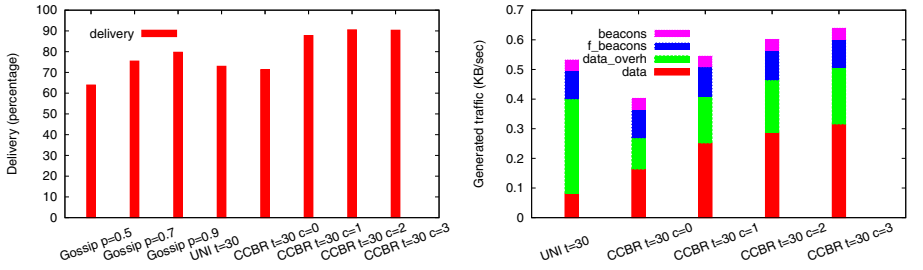


Fig. 2. The results measured in the default scenario

are added to such beacons once every three of them) and the initial number of credits; for Gossip the forwarding probability; for Uni the beaconing interval.

This very extensive analysis resulted in a large body of data (and graphs), which allowed us to examine all the aspects of CCBR and how it performs w.r.t. Gossip and Uni. On the other hand, the limited space available does not allow to report here all the tests we have done. Next sections discuss a subset of them, those we found most relevant[2].

3.1 The Default Scenario

Our default scenario reflects the situation of a set of persons or animals moving around in a limited area and being monitored by both mobile and fixed sinks. In particular, we consider an area of 0.5 Km^2, 50 sensors, and three sinks. Nodes (i.e., all the sensor nodes and two sinks over three) move according to a random waypoint mobility model with a speed between 1 and 2 m/s and a stop period of up to 10s. Sensors send a message every 10s, while the interests of each sink have 10% of chances to match each of the published messages, which means that 28% of the messages sent has to be delivered to at least one of the three sinks (i.e., Uni and CCBR only have to deliver one message every 36 seconds, on average).

Figure 2 (left) shows the delivery we measured for every protocol in this scenario. CCBR with zero credits, Uni, and Gossip with $p = 0.7$ provide similar results, correctly delivering between 72% and 74% of messages, while CCBR with the retransmission mechanism in place (i.e., when credits are greater than zero) provides the best performance, reaching a very good 90% of delivery with two credits. To put these numbers in context, we notice that the range of communication resulted, from our model, around 100m in absence of any other communication (i.e., no interferences), being much shorter when several nodes transmit data at the same time, as it happens in our scenario. As a result, our default

[2] The reader is warned that, even if we did not plot the confidence intervals in the graphs below (to avoid cluttering them), we took them into consideration. In particular, we run each simulation several times, varying the seeds of the random number generators we used in our models, until the size of the 95% confidence interval of the sample mean we measured was below 5% of the mean itself.

density of 100 nodes per Km^2 results in periodic partitions of the network, which explains why none of the protocols we considered reaches a 100% of delivery.

As a further observation, it could appear strange how Gossip with a high probability of forwarding (e.g., $p = 0.9$) does not provide the best delivery. To understand why this happens we observe (graph not reported for space reasons) that Gossip generates 15 times more traffic than CCBR and Uni: 7.7 KB/s for Gossip with $p = 0.9$ vs. 0.53 KB/s for Uni, thus incurring in a number of collisions and interferences that strongly limits its capacity of delivering messages.

A more detailed analysis of the traffic (measured at the physical layer) generated by CCBR and Uni in the default scenario is provided by Fig. 2 (right). First, we observe how the traffic generated by CCBR increases less than linearly with credits: a very positive result that proves the efficiency of the CCBR's retransmission mechanism. We also notice how beacons, including those carrying filters (i.e., `f_beacons` in figure), contribute around one fourth of the overall traffic. This is reasonable since in the default scenario each sensor produces, on average, one message every 36 seconds that is worth transmitting to at least one of the sinks, while the three sinks emit one beacon every 30 seconds which flood the network (albeit with the efficient "delay and cancel" mechanism explained above) to keep tables up to date, despite of mobility.

The last thing worth observing is how the total traffic generated by Uni is much greater than that generated by CCBR with zero credits, which performs comparably w.r.t. delivery. This is counterintuitive, since in the default scenario, in which most of the messages are addressed to a single sink, a routing based on unicast communication and shortest path tree forwarding, like Uni, should generate the least traffic to deliver messages. To understand why this happens, in Fig. 2 (right) we distinguished between the traffic generated by each protocol and sent to the MAC layer ("data" in figure), and the overhead generated by the MAC and physical layers ("data overhead" in figure). If we look at the first number alone, we see how Uni produces much less traffic than CCBR at the "routing" layer (above the MAC). Unfortunately, it incurs in a lot of overhead at the MAC and physical layers. This can be explained by remembering that 802.15.4 is a "reliable" MAC, using acknowledgements and multiple retransmissions to correctly deliver unicast packets. In presence of mobility this approach is detrimental, since the MAC wastes a lot of bandwidth in trying to reach nodes that went out of range. Conversely, CCBR uses broadcast at the MAC layer, which is unreliable but incurs much less overhead. This result supports our belief (see Sect. 2.2) that using broadcast and leaving the decision of forwarding to the receiver of a packet, not to the sender, whose routing tables may be outdated, is the right choice in presence of mobility.

3.2 The Impact of Credits and Beaconing Interval vs. Speed

CCBR was especially designed to support mobile scenarios, so it is important to see how it behaves under different patterns of mobility and how the beaconing interval and the number of credits impact its performance. In particular, while keeping all the other parameters as in the default scenario, we considered three

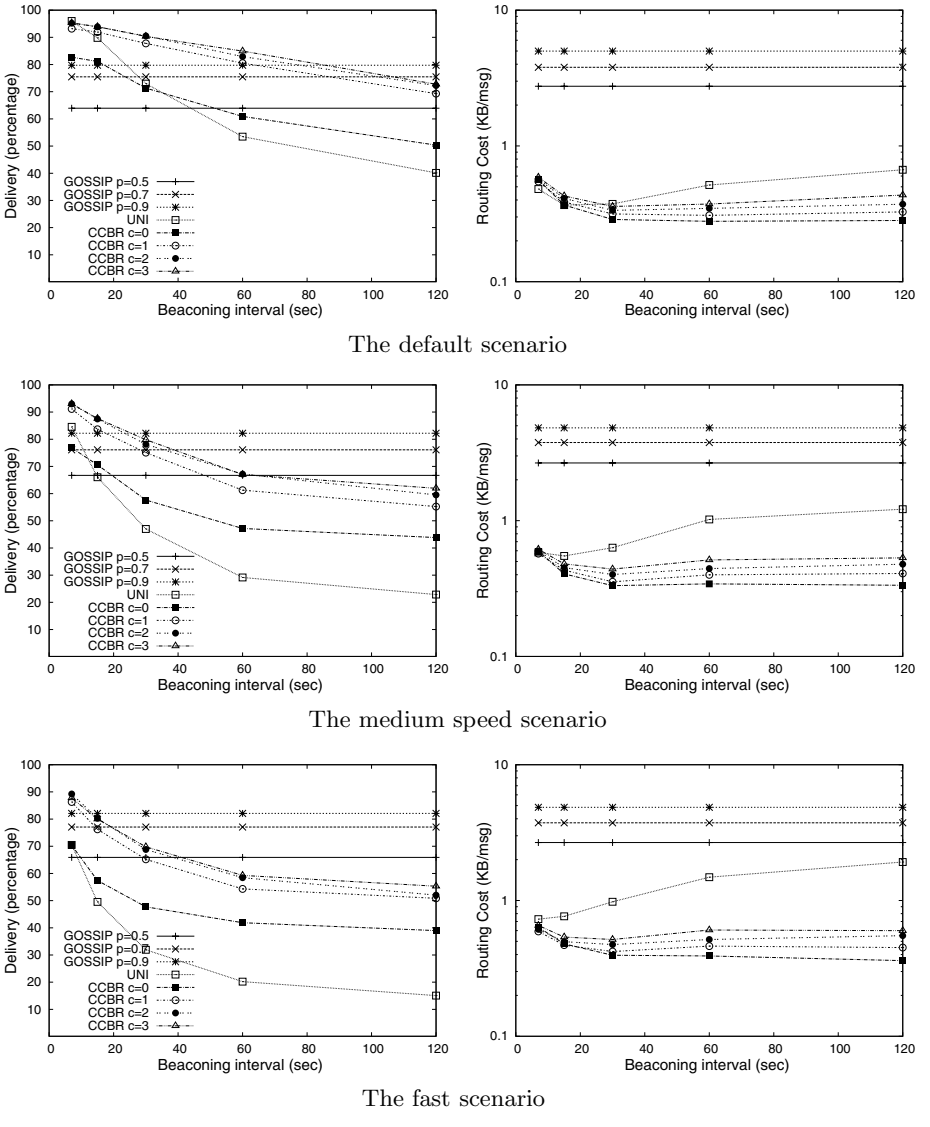


Fig. 3. The impact of speed and beaconing interval on results

levels of mobility: the default one, a "medium speed" scenario with nodes moving between 3 and 5 m/s and stopping for at most 2 seconds, and a "fast" scenario with nodes continuously moving at a speed between 5 and 10 m/s.

In Fig. 3 we plot the delivery and the *routing cost*[3], measured in KByte of traffic (generated at the physical layer) per delivered message, in these three scenarios. First we may notice how, in all scenarios, Uni is the protocol which

[3] Notice how the graphs plotting the routing cost are in logarithmic scale.

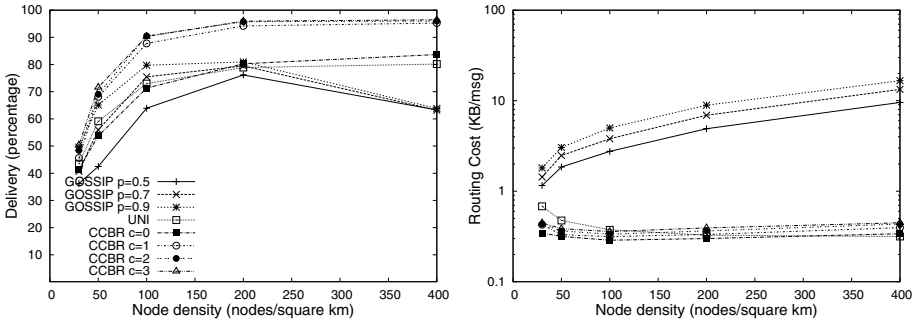


Fig. 4. The impact of node density on results

suffers more when the beaconing interval grows. On the contrary, the delivery of CCBR, even with zero credits, i.e., with the retransmission mechanism disabled, decreases much more slowly.

The efficacy of the retransmission mechanism can be measured by looking at CCBR with one or more credits. Its delivery is much better w.r.t. the case with zero credits, with the impact of the beaconing interval becoming less and less relevant as the number of credits grows. In the default scenario, CCBR with two credits provides 75% of delivery even with one beacon every 120 seconds, while in the hardest scenario of mobility one beacon every 30 seconds is enough for CCBR with two credits to provide a reasonable 70% of delivery. In the same situation the delivery of Uni is below 33%.

The graphs on the right in Fig. 3 show how CCBR, in every scenario of mobility we considered, is much better than Uni at keeping the routing cost constant while the beaconing interval grows. This, again, shows how a sender-based, unicast approach to routing, with automatic retries at the MAC layer, is not well suited to propagate packets in presence of mobility, i.e., when the correctness of the routing tables cannot be guaranteed. The same graphs also show how the number of credits has a minimal impact on the routing cost, a measure of the efficiency of the retransmission mechanism.

3.3 Density of Nodes and Area of the Network

Figure 4 shows the impact of a changing density of nodes when all the other parameters remain fixed at their default values. We observe that all protocols except Gossip increase their delivery as the density grows. Gossip performs better when density grows but only up to a certain limit (200 nodes per Km^2). After that point collisions become an issue even for the lowest forwarding probability we tested.

To compare the three protocols we may notice how CCBR with zero credits, Uni, and Gossip with $p = 0.7$ provide similar performance up to 200 nodes per Km^2, with CCBR and Uni also performing similarly up to the maximum density. On the other hand, the delivery tells only one side of the story. If we look at the routing cost we notice how Gossip with $p = 0.7$, while delivering

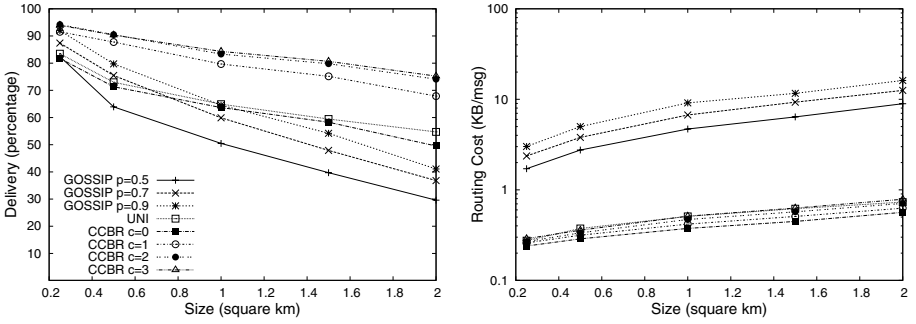


Fig. 5. The impact of an increasing area on results

the same percentage of messages of CCBR and Uni, uses much more bandwidth (the graph is in logarithmic scale). CCBR with zero credits and Uni incurs in very similar costs, with the former performing better when the density is low.

Even in this case, the retransmission mechanism proves its efficacy and efficiency. Indeed, CCBR with one or more credits outperforms all the other protocols at every density, while showing a routing cost that is only slightly greater than that of CCBR with zero credits and Uni.

Finally, we notice how the routing cost for CCBR is very marginally influenced by the density of the network. This is a very positive result for CCBR, that must be ascribed to the "delay and cancel" mechanism of forwarding, which minimizes useless retransmissions when the density grows.

Another interesting question to answer is how the performance of the different protocols changes when the area of the network grows (at a constant density of nodes). What we expect is an increase in the cost to deliver each packet, since the number of hops to travel increases, and this is confirmed by our tests (see Fig. 5), with CCBR and Uni decreasing their efficiency less than Gossip.

As for delivery, it decreases much more slowly when CCBR and Uni are adopted instead of Gossip, with CCBR outperforming Uni especially when the retransmission mechanism is used (i.e., with one or more credits). In a field of 2 Km^2 CCBR with two credits delivers 36% more messages than Uni, i.e., from 55% to 75% of delivery. The same figure also shows how the larger is the network the better it is to use more credits. This can be explained by observing that the more hops a packet has to travel the more credits it requires to overcome possible problems (i.e., local minima).

3.4 Static Scenario and Multiple Recipients Per Message

After seeing how CCBR behaves in the mobile scenarios it was designed for, we are interested in seeing how it performs in a static scenario, and how it compares with Uni, which should operate at best when nodes are firm. Since CCBR was developed to support multi-sink scenarios, we are interested in seeing how the number of message recipients impacts its performance in this stationary scenario.

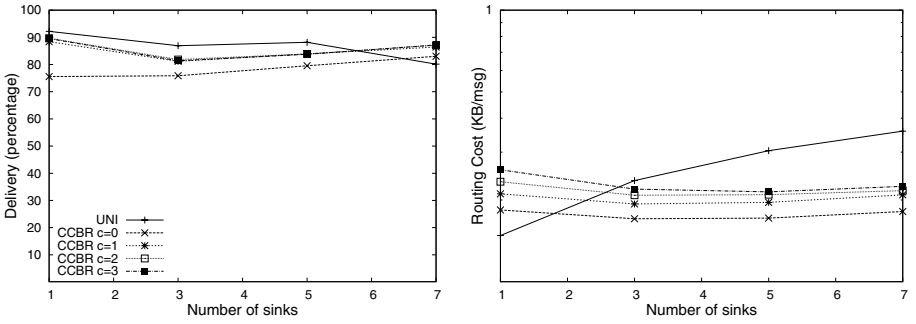


Fig. 6. The results for a static scenario (firm nodes)

Accordingly, we measured the performance of CCBR and Uni with firm nodes, letting each message go to every sink and progressively increasing the total number of sinks. Figure 6 shows the results we had. As we expected, Uni provides the best delivery in this scenario, with CCBR being very close and surpassing Uni when the number of sinks grows over a certain limit (at that point the contention on the channel to deliver the same packet to multiple sinks becomes an issue for Uni). What is even more positive is to observe how the cost of routing for CCBR is initially greater than that of Uni, but becomes smaller as soon as the number of receivers of each message grows. This shows that we centered our second goal: CCBR is capable of optimizing routing when the same message has to be delivered to more sinks.

As a final remark, by comparing the behavior of CCBR and Uni in the single sink, static scenario (the hardest one for CCBR), we may evaluate the efficiency of the "delay and cancel" mechanism. Number at hands, we measured (for CCBR with zero credits) 24% more traffic than Uni. This means that once every four hops the delay and cancel mechanism "fails". A good result considering that this mechanism has been designed to provide the robustness and efficiency required for very different scenarios, i.e., those including mobility and multiple sinks.

4 Related Work

While fully mobile WSNs have been rarely considered so far, the case of mobile sinks in a stationary WSN is not new [10,11,12]. Sink mobility has been also considered as a way to transform the problem of routing data toward the sinks into the problem of routing sinks towards the data [13,14,15,16].

The case of WSNs involving mobile sensors, like those deployed to monitor animals [17,18], is also not new. Unfortunately, the routing protocols proposed in this area [19,20,21,22] focused on strongly disconnected scenarios, where the very low density of nodes requires mechanisms typical of delay-tolerant networks. As mentioned, we consider different scenarios in which sink reachability is more the rule than the exception, thus allowing real-time monitoring of critical situations.

These scenarios are typical of Mobile Ad-hoc NETworks (MANETs), indeed CCBR is based on our previous experience in developing content-based publish-subscribe routing protocols for MANETs [4,5,6,7]. However a direct application of these and other content-based routing proposals for MANETs (e.g., [23]) to WSNs is hard because of sensors' tight resource constraints.

To the best of our knowledge CCBR is the first data-aware (i.e., content-based) routing protocol especially designed for mobile WSNs. Other protocols adopting a similar, data-aware model, such as Directed Diffusion (DD) [2], GRAB [24], and TinyCOPS [3] have considered fixed networks as their reference scenario [25]. An evaluation of these protocols in mobile scenarios will be beyond their intended scope, even unfair [26], since they only consider (i) long-term faults with a frequency in the range of hours or days, due to slow fading and node failures, (ii) short-term faults with frequency in the order of μs due to transmission errors and collisions. As an example, DD handles the former with path repairing mechanisms, while delegates the latter to unicast, reliable MACs. Mobility dynamics, being in the order of seconds, sits almost in the middle of these two extremes, and cause faults that are too frequent to be handled by repair mechanisms and too long lasting to be handled by retransmission. Moreover, as also noted in [27] and confirmed by our simulations, unicast and reliable MACs aggravate the situation, by interpreting faults due to mobility as transmission errors, causing useless delays and wasting bandwidth and energy.

GRAB strives for higher robustness with respect to both short and long-term faults by abandoning sender-based, unicast forwarding in favor of link layer broadcast primitives and receiver-based forwarding. Differently from CCBR, however, to increase robustness it does not suppress redundant retransmissions: each node within a certain, progressively reducing, distance from the sink (resulting in a lens-shaped area) retransmits the message. Unfortunately, the distance field used by GRAB for taking receiver-based forwarding decisions rapidly deteriorates when nodes move. As before, while GRAB approach is to rebuild the cost field in those situations, it lacks mechanisms to mask faults induced by mobility, such as the CCBR retransmission and credits mechanism.

Other mechanisms adopted by CCBR are receiver contention, channel overhearing and RSSI estimations. They exploit WSNs peculiarities to allow distributed routing decisions with minimal state and communication. Other proposals in WSNs use similar mechanisms. Receiver contention is used by geographic routing [28,29,26], opportunistic routing [30], data dissemination [31], and cooperative diversity [32] schemes to select the best forwarder or relay with minimal overhead. CCBR uses this approach with a different goal: to select efficient routes leading to multiple destinations in spite of unreliable routing information resulting from mobility. Channel overhearing is used in geographic routing to trigger void avoidance phases, while CCBR (using hop-counts) does not have voids but uses instead overhearing to trigger its local minima escape mechanism. Finally, exploiting RSSI to quickly establish distances was already proposed in [33].

5 Conclusion

While the typical application for WSNs is in environmental monitoring, different scenarios are also possible. In particular, in the WASP project financed by the EU Commission we are considering scenarios like controlling animals in a farm or monitoring elder people in hospices, which involve mobile nodes and multiple sinks, while continuous connectivity is guaranteed by the small area in which sensors move (e.g., the field in which cattle move) or by the presence of fixed sensors that may act as forwarders (e.g., in a house).

The CCBR protocol we described in this paper provides a context and content-based routing layer especially tailored to such scenarios. On top of this layer, it becomes easy to develop several communication paradigms, from publish-subscribe to continuous queries ala TinyDB. Our simulations show that the mechanisms used by CCBR are very effective in providing good delivery with a low cost of routing, which potentially implies a low power consumption.

With respect to the issue of power consumption, other partners of the WASP project developed a MAC protocol optimized for the kind of broadcast communication adopted by CCBR. It guarantees low power drain both for sending and for receiving broadcast packets by extensively using advanced duty cycling mechanisms. We are currently implementing CCBR on top of the first release of this MAC to measure power consumption on real nodes.

As a further work in this area, we are interested in investigating how to add "in network processing" capabilities to CCBR, to allow sinks to specify some aggregation function, letting CCBR decide where to aggregate data. A very complex task in mobile scenarios like those we target.

References

1. Carzaniga, A., Wolf, A.L.: Content-based networking: A new communication infrastructure. In: König-Ries, B., Makki, K., Makki, S.A.M., Pissinou, N., Scheuermann, P. (eds.) IMWS 2001. LNCS, vol. 2538, pp. 59–68. Springer, Heidelberg (2002)
2. Intanagonwiwat, C., Govindan, R., Estrin, D., Heideman, J., Silva, F.: Directed diffusion for wireless sensor networking. Trans. on Netw. 11(1), 2–16 (2003)
3. Hauer, J.H., Handziski, V., Köpke, A., Willig, A., Wolisz, A.: A component framework for content-based publish/subscribe in sensor networks. Wireless Sensor Networks, 369–385 (2008)
4. Mottola, L., Cugola, G., Picco, G.P.: A self-repairing tree topology enabling content-based routing in mobile ad hoc networks. IEEE Trans. on Mobile Computing 7(8), 946–960 (2008)
5. Baldoni, R., Beraldi, R., Querzoni, L., Cugola, G., Migliavacca, M.: Content-based routing in highly dynamic mobile ad hoc networks. Int. Journ. of Perv. Comp. and Comm. 1(4), 277–288 (2005)
6. Costa, P., Migliavacca, M., Picco, G.P., Cugola, G.: Epidemic algorithms for reliable content-based publish-subscribe: An evaluation. In: ICDCS, pp. 552–561 (2004)

7. Picco, G.P., Cugola, G., Murphy, A.L.: Efficient content-based event dispatching in the presence of topological reconfiguration. In: ICDCS, pp. 234–243 (2003)
8. OMNeT++ Web page, http://www.omnetpp.org
9. Mobility Framework for OMNeT++ Web page, http://mobility-fw.sourceforge.net
10. Luo, H., Ye, F., Cheng, J., Lu, S., Zhang, L.: Ttdd: Two-tier data dissemination in large-scale wireless sensor networks. Wireless Networks 11(1-2), 161–175 (2005)
11. Kim, H.S., Abdelzaher, T.F., Kwon, W.H.: Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In: SenSys (2003)
12. Hwang, K.-i., In, J., Eom, D.-S.: Distributed dynamic shared tree for minimum energy data aggregation of multiple mobile sinks in wireless sensor networks. In: Römer, K., Karl, H., Mattern, F. (eds.) EWSN 2006. LNCS, vol. 3868, pp. 132–147. Springer, Heidelberg (2006)
13. Shah, R., Roy, S., Jain, S., Brunette, W.: Data mules: Modeling a three-tier architecture for sparse sensor networks. In: IEEE SNPA Workshop (2003)
14. Somasundara, A., Kansal, A., Jea, D., Estrin, D., Srivastava, M.: Controllably mobile infrastructure for low energy embedded networks. IEEE Transactions on Mob. Comp. 5(8), 958–973 (2006)
15. Chatzigiannakis, I., Kinalis, A., Nikoletseas, S.: Efficient data propagation strategies in wireless sensor networks using a single mobile sink. Comput. Commun. 31(5) (2008)
16. Ammari, H.M., Das, S.K.: Promoting heterogeneity, mobility, and energy-aware voronoi diagram in wireless sensor networks. IEEE TPDS 19(7), 995–1008 (2008)
17. Bonnet, P., Leopold, M., Madsen, K.: Hogthrob: towards a sensor network infrastructure for sow monitoring. In: DATE (2006)
18. Butler, Z., Corke, P., Peterson, R., Rus, D.: Dynamic virtual fences for controlling cows. In: Experim. Robotics IX. Springer Tracts in Adv. Rob. Springer, Heidelberg (2006)
19. 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, San Jose, CA (October 2002)
20. Pasztor, B., Musolesi, M., Mascolo, C.: Opportunistic mobile sensor data collection with scar. In: MASS (2007)
21. Henriksson, D., Abdelzaher, T., Ganti, R.: A caching-based approach to routing in delay-tolerant networks. In: ICCCN 2007, August 2007, pp. 69–74 (2007)
22. Luo, T.L., Huang, C., Abdelzaher, Stankovic, J.: Envirostore: A cooperative storage system for disconnected operation in sensor networks (2007)
23. Petrovic, M., Muthusamy, V., Jacobsen, H.A.: Content-based routing in mobile ad hoc networks. In: MobiQuitous, pp. 45–55 (2005)
24. Ye, F., Zhong, G., Lu, S., Zhang, L.: Gradient broadcast: a robust data delivery protocol for large scale sensor networks. Wirel. Netw. 11(3), 285–298 (2005)
25. Bokareva, T., Bulusu, N., Jha, S.: A performance comparison of data dissemination protocols for wireless sensor networks. In: GlobeCom Workshops 2004, November -3 December 2004, pp. 85–89. IEEE, Los Alamitos (2004)
26. He, T., Blum, B.M., Cao, Q., Stankovic, J.A., Son, S.H., Abdelzaher, T.F.: Robust and timely communication over highly dynamic sensor networks. Real-Time Syst. 37(3), 261–289 (2007)
27. Heissenbüttel, M., Braun, T., Wälchli, M., Bernoulli, T.: Evaluating the limitations of and alternatives in beaconing. Ad Hoc Netw 5(5), 558–578 (2007)
28. Zorzi, M., Rao, R.R.: Geographic random forwarding (geraf) for ad hoc and sensor networks: Multihop performance. IEEE Trans. Mob. Comput. 2(4), 337–348 (2003)

29. Heissenbüttel, M., Braun, T., Bernoulli, T., Wälchli, M.: Blr: Beacon-less routing algorithm for mobile ad hoc networks. Comp. Comm. 27(11), 1076–1086 (2004)
30. Biswas, S., Morris, R.: Exor: opportunistic multi-hop routing for wireless networks. In: SIGCOMM, pp. 133–144 (2005)
31. Mastrogiovanni, M., Petrioli, C., Rossi, M., Vitaletti, A., Zorzi, M.: Integrated data delivery and interest dissemination techniques for wireless sensor networks. In: GLOBECOM (2006)
32. Bletsas, A., Khisti, A., Reed, D.P., Lippman, A.: A simple cooperative diversity method based on network path selection. IEEE JSAC 24(3), 659–672 (2006)
33. Dutta, P., Culler, D., Shenker, S.: Procrastination might lead to a longer and more useful life. In: 6th Workshop on Hot Topics in Networks (HotNets VI) (2007)

Dynamic Source Routing versus Greedy Routing in a Testbed Sensor Network Deployment

Hannes Frey[1] and Kristen Pind[2]

[1] University of Paderborn
hannes.frey@uni-paderborn.de
[2] University of Southern Denmark
kristen@imada.sdu.dk

Abstract. We present our findings of an empirical performance comparison between dynamic source routing and greedy routing in a testbed wireless sensor network deployment. The environment is based on a grid ranging between 2×2 to 7×7 battery operated Tmote Sky sensor nodes. We briefly sketch the protocol implementations and the experimental setup used in this work. We also discuss how we managed such larger set of nodes in a convenient way. We hope that some of the lessons we learnt may be useful to others doing such kind of testbed experiments as well. Finally, we discuss our findings from the measurement data we have gathered.

1 Introduction

This work describes our recent advances in providing real world measurements of basic wireless data communication primitives. At the time of writing we have been focusing on end-to-end communication between a source and destination node. We believe that a basic understanding of such basic primitives will as well provide insights in more advanced sensor network specific data collection primitives. Moreover, sensor networks are not only about data collection trees. For instance, a data sink issuing a sensor request into a specific geographic region, a data sink issuing a request to a certain area identified by a specific node, or a data source trying to connect to a specific actuator are just a few examples which show that sensor networks require basic unicast communication primitives as well.

Data communication protocols can be classified in topology based and geographic ones. Surveys can be found in [1] and [2], respectively. Topology based communication requires either proactive or reactive exchange of global information before message forwarding from source to destination can take place. Geographic message communication in contrast does not require such global message exchange. In such routing mechanism it is assumed that each node knows its current physical location. Based on this additional information a routing decision can be performed in a pure *localized* manner, i.e., a forwarding decision requires only information about the position of the current node, its immediate neighbors, and the message destination. It is believed that such localized protocols

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 86–101, 2009.
© Springer-Verlag Berlin Heidelberg 2009

provide scalable solutions, that is, solutions for wireless networks with an arbitrary number of nodes. Future sensor networks, for instance, might consist of thousands or ten thousands of nodes.

Research on localized algorithms – in particular the comparison of localized versus non-localized approaches – has focused mainly on theoretical investigations and simulation. While both form an important aspect in getting a thorough understanding on what is going on, a critical part has widely been neglected so far. It is of paramount importance to perform the comparison "localized versus non-localized" in the domain where the protocols are supposed to be applied. Localized algorithms are not supposed to run in Ns2, Omnet, or Qualnet. They are supposed to run on real hardware platforms like MicaZ, or Tmote Sky nodes.

What follows is just a drop in a bucket, to provide some numbers of real-world measurement for comparing the domain of localized versus non-localized data communication. We focused at first on two protocols. We selected greedy routing as a representative of the localized domain and dynamic source routing as a representative of the non-localized (or topology-based) domain. The protocols and how we implemented them are described briefly in Section 2. Then we describe our experimental setup in Section 3. The results are presented in Section 4. We conclude our work and provide some future research directions in Section 5.

2 Protocol Implementations

We implemented dynamic source routing and greedy routing in nesC [3] under the TinyOS 2.0 execution environment [4]. Wireless communication between immediate neighboring sensor nodes and wired serial communication between sensor node and host computer is implemented on the basis of Active Messaging [5]. The maximum allowed message size was fixed to 64 bytes payload. We used the standard IEEE 802.15.4 MAC protocol provided by the mixed software/hardware CC2420 radio stack in TinyOS, without changing any settings or implementations.

The following provides a brief outline on both implemented protocols and motivates the rationale behind our implementation decisions. For both variants we investigated in our studies a version with and one without link layer acknowledgement. We used the link layer acknowledgement realized in the CC2420 radio stack. When using the acknowledgement feature, we set the maximum number of retransmissions to 3 and the retransmission delay to 0 msec.

2.1 Dynamic Source Routing

The Algorithm Principle. Dynamic source routing [6], DSR in short, basically works as follows. The source node sends out a route request message RREQ which is repeated by all intermediate nodes once they receive it for the first time. Additional RREQ receptions are just ignored. In other words, the RREQ message is just flooded over the entire network.

Every RREQ message copy stores the sequence of nodes it has visited so far. In this way several copies of the RREQ message will eventually reach the

destination node which can then pick up one of these messages to send a route reply RREP back along the reverse of the path stored in the RREQ messages. To achieve this, the RREP message just stores the sequence of nodes which have to be visited from source to destination. Once the message arrives at the source node, the reverse of the path stored in the RREQ message can finally be used to send data towards the destination node.

Implementation. Our DSR implementation provides a timeout which can be used to invalidate the current route form source to destination. This means when using this timeout a new route discovery will be initiated periodically. However, this is just an optional feature which is useful to adapt proactively to a dynamically changing network topology; due to mobility or any kind of fading effects. We used this feature in those kinds of measurements where we investigated delay induced by DSR route recovery under dynamic network topologies.

Finally, when using link layer acknowledgments our DSR implementation provides error notification at the source in case of path breakage. A node which realizes that the link to the next hop along the path is broken, will send an ERR message along the reverse path to the source node. When triggered by the next message from the upper layer, the source node will send out a next RREQ in order to find a new path from source to destination.

2.2 Greedy Routing

The Algorithm Principle. The greedy routing principle [7,8,9] works as follows. Each node is supposed to know its own location in terms of some given coordinate system. Messages store the location of the destination node as well. A current forwarding node has to acquire the position of its one hop neighbors and then choose that next hop neighbor whose location is the best one with respect to the current node's location, the final destination's location, and the localized metric being applied. If the current node is the best one, the message will be dropped.

Implementation. In greedy routing each node is supposed to know the locations of the nodes in its vicinity. In this work we focused on the so called beacon less greedy routing variant [10,11,12]. However, we implemented the following active neighbor selection variant. As long as a node has not to forward anything, it is not supposed to know its neighbors. As soon as a message gets in, the neighbors have to be determined first. The node will broadcast a neighbor discovery message NDSC first. Any neighbor node receiving a NDSC message will send a unicast reply message NREP which includes the node's ID, its location, and a value indicating the quality of the link from the NDSC originator to the NDSC recipient. On reception of a NDSC message, the latter value is immediately requested by the NDSC recipient from the CC2420 TinyOS module. Finally, to keep the number of collisions low, a node replying with a NREP message will wait a random time between 0 and 50 msec before sending the NREP back to the NDSC initiator.

In a static topology, neighbor discovery needs to be performed only once. As soon as the first message was handled, the other messages coming in can use the already determined set of one hop neighbor nodes. In a highly dynamic scenario, however, this set will be outdated and a new neighbor discovery gets necessary. To support any topology properties between those two extreme cases we implemented a timeout based scheme where the neighbor set is invalidated periodically. Thus, message handling within one period requires neighbor discovery only once. The timeout period can be set as a protocol parameter.

In addition, when using link layer acknowledgements our greedy implementation supports a simple error recovery mechanism. On failure notification from the link layer, the entire neighbor list is erased. When a message has to be forwarded from a node with an empty neighbor list, a new list will be set up by issuing a NDSC message and waiting 70 msec for collecting NREPs and resuming the next greedy forwarding step.

In this work we focus on two basic greedy routing metrics, a *distance based* and a *link quality based* one. In the distance based metric the next forwarding step is the node which minimizes the distance to the destination. We refer to this method as Greedy-dist in the following. In the link quality based variant, the node which maximizes the link quality indicator value will be selected. We refer to this variant as Greedy-lqi in the following. In both variants, however, in order to prevent routing loops, only nodes which are closer to the destination than the current node are used.

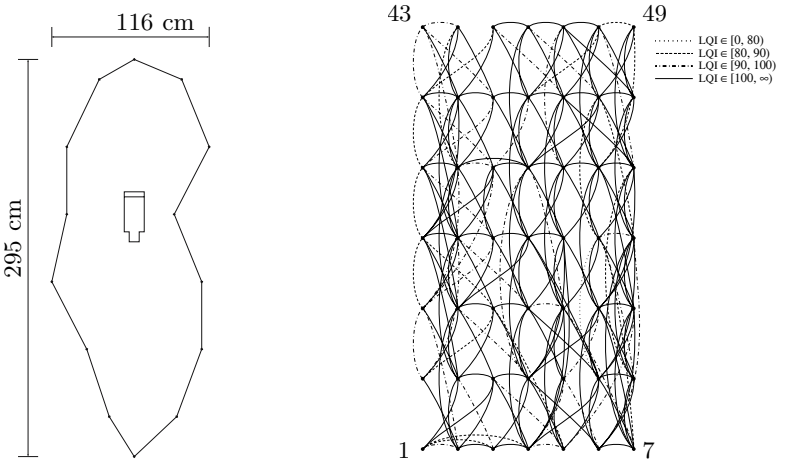
Location information is an important ingredient of any localized data communication protocol. Here we focused on the routing aspects only. We implemented a TinyOS module which provides an interface to request a node's position. The current implementation just returns a preconfigured location value which is set in the phase of mote flashing.

3 Experimental Setup

The following described experiments were performed on a grid of battery operated Tmote Sky sensor nodes (see `www.sentilla.com`). We varied the grid size between 2×2 to 7×7 nodes. The network was deployed indoors in a single room. Nodes were located on the ground and have all been oriented such that their antennas were pointing in the same direction. No obstacles have been placed in between them.

To have a reasonable multi-hop experimental scenario on a small experiential area, transmission power of each node was set to 1 (out of a range between 0 and 31). We investigated that a transmission power 0 rendered many message losses due to weak signal strength. A signal strength above 1 kept too many nodes connected thus rendering the topology almost fully connected.

With transmission power set to 1, we were interested in placing nodes with about the same density in both, column and row direction. To achieve this, we determined a rough estimate of the radiation pattern of the sensor nodes (see Fig. 1a). We used one node continuously transmitting short packages whose



(a) Radiation pattern. **(b)** Network topology.

Fig. 1. Setting up the experiment

correct reception was measured in a second sensor board which we moved away until the first messages got lost. In this way we determined the depicted reference points. While we do not claim that the depicted polygon reflects a precise radiation pattern, the shape however gave us a rough picture of the directionality of the sensor board's antenna.

Our measurement suggested an antenna radiation being in column direction between 2 to 3 times larger than in row direction. In order to get about the same density in both column and row direction, we decided on row distances two times larger than column distances. Finally we placed nodes such that immediate column and row neighbors will likely see each other. To achieve this we selected node distances such that neighbor nodes are located about on the half between a node and the boundary of its radiation polygon. According to this, as depicted in Fig. 1b, nodes were equally aligned and placed on a grid with 25 cm grid width and 50 cm grid height.

The links in the figure depict for one example setup the node's capabilities in reaching another node with a reception probability above 75%. This means, out of 100 transmitted messages, at least 75 were received. Depicted are bidirectional links only, i.e., in both direction at least 75 messages were received. Links are further distinguished according to the average link quality indicator values provided by the CC2420 module. LQI values may differ for the communication direction along one link. In this graphical representation we show the minimum over both LQI values of each link.

The grid structure (see Fig. 1b) was also used to determine the nodes' IDs and relative coordinates. Node IDs just count the nodes from the bottom left to the top right corner. Greedy routing does not require absolute node positions. Any virtual coordinate system which supports computing distances between nodes is sufficient. We have set the node positions according to their location in the

grid, with the node in the bottom left corner having position $(1, 1)$ and the node in the top right corner having position $(7, 13)$. More precisely, a node on grid position (i, j) has position $(i, 2j)$ in our relative coordinate system.

3.1 Controlling the Experiments

To control the experiment, node 1 and node 49 have been attached via USB to a notebook computer. We implemented a Java application to generate the traffic, and collect measurement data from the final destination node.

We implemented a TinyOS module listening for control messages coming from the USB port. Two classes of control messages exist. One class of control message is flooded through the entire network to reach all nodes. This way we are able to manage the whole set of nodes from a single notebook computer without the need to touch a single sensor after flashing. Control messages out of this class implemented so far enable selection between Greedy-dist, Greedy-lqi, and DSR and support resetting these protocols to an initial state.

The second class of control messages is not flooded but only addressed to one of the nodes attached to the notebook computer directly. So far we implemented a single instance of this kind of message in order to inject data communication tasks into the network. More precisely, the Java application tells the attached source node to send a data packet periodically to the destination node.

3.2 Measuring Time

Due to high jitter we observed for the USB communication between sensor nodes and notebook computer, we decided on time measurement in the source and destination sensor nodes directly. We used the difference Δ between send time at the source and receive time at the destination. To determine the duration of one routing message to get from source to destination we add the clock offset between source and destination to Δ. To determine that clock offset, before each measurement source and destination node perform 100 direct message exchanges at full signal strength to determine the average over time at source node minus time at destination node minus half of the round trip delay of direct transmission. We observed a very predictable round trip delay with a 4 msec standard deviation which provided us some confidence in computing the clock offset between source and destination in that way.

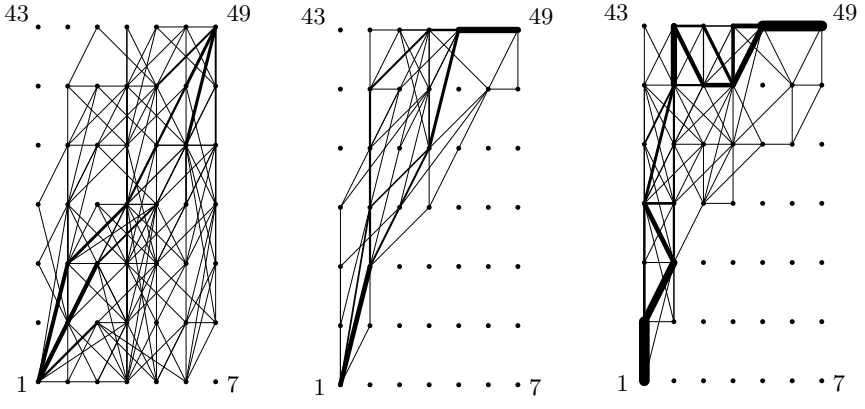
4 Results

In the following graphs we compare the performance of DSR and greedy routing in terms of *delay*, *delivery ratio*, *hop count*, and *data rate*. Delay is the time interval between packet transmission at the source and packet reception at the destination. The delivery ratio is the number of packets successfully delivered at the destination over the total number of packets sent by the source. Hop count is just the number of nodes visited by a packet when it gets delivered at the destination. The data rate is the amount of data which is successfully received at the destination node per time unit.

4.1 Intuition from Path Accumulation Diagrams

In order to get a first intuition on the performance to be expected, Fig. 2 shows the *path accumulation diagram* for the three considered routing protocols. The diagrams are a pictorial representation of (1) the paths visited by a protocol for a given number of independent routing tasks, and (2) the frequency of the links taken along these paths. Every depicted link means that this link was used at least once. The thicker a link the more frequent this link was taken. The figure was obtained from 500 independent routing tasks. In each routing task a single message was sent from node 1 to node 49.

The diagrams show that the paths taken by DSR involve more network links than the Greedy variants. However, there is a concentration on the links following the diagonal edges from the source node to the destination node. The paths taken by the Greedy variants involve less nodes of the network. In both cases the paths concentrate on the upper diagonal of the experimental setup. We explain this by the fact that in this setting Greedy routing favors nodes located in the north south direction over nodes located in the east west direction. Due to the directionality of the sensor nodes antennae and due to the alignment of nodes, a node in north south direction is easier to reach than a node in east west direction. Moreover, the next row is providing more progress towards the destination than the next column. Thus, for both greedy routing variants it is more likely to select a next hop node along the column than a node along the row. DSR in contrast is flooding the entire network thus it is becoming more likely that all nodes in the network sometimes get involved in a routing path. Finally, both for, DSR and Greedy-dist several long links bypassing a whole row or column can be observed. This is different to Greedy-lqi where this happens less frequently. Here, the most frequently used links are the shortest possible ones.



(a) DSR. **(b)** Greedy-dist. **(c)** Greedy-lqi.

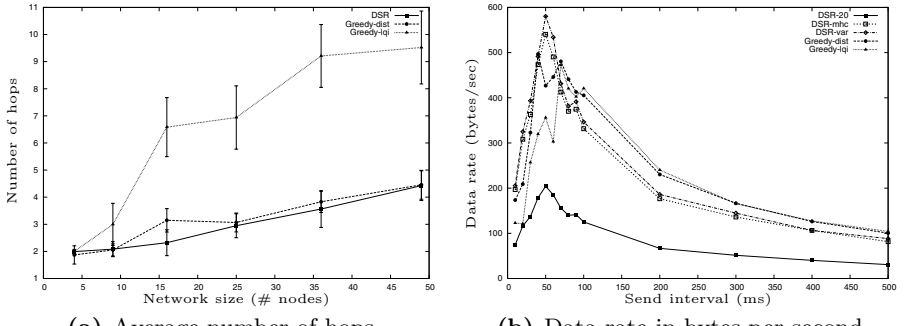
Fig. 2. Path accumulation diagrams

4.2 Evaluating the Static Case

We were interested in the protocol behavior in terms of hop count, data rate, delay, and delivery ratio, when network size increases. Experiments have been performed for grid sizes ranging from 2×2 to 7×7. Data packets were injected at the grid's bottom left corner, i.e., node 1, and routed to the grid's top left corner, e.g., node 49 in case of the 7×7 grid.

Hop Count and Data Rate. Fig. 3a depicts the average number of hops performed by DSR, Greedy-dist, and Greedy-lqi over increasing field sizes. We considered only the case with link layer acknowledgments turned on. Each measurement point resulted from 500 independent routing tasks, with each routing task sending out a single message. The hops taken by the successful messages have been averaged. The figure shows in numbers what intuitively is suggested by the path accumulation diagrams depicted in Fig. 2. Greedy-lqi selecting high quality links, favors short links over long ones. DSR selecting the first route request arriving at the destination, implicitly selects paths with a low number of hops. The same applies for Greedy-dist. Selecting the next hop node closest to the destination results in a short hop path. In our simulation setting, Greedy-lqi resulted in path lengths being slightly more than twice the path lengths of DSR and Greedy-dist. In other words, the average length of best quality links selected by greedy-lqi is roughly half of the length of the link providing the largest advance towards the destination node.

Next, we discuss the maximum data rate which can be supported by the different routing protocols. Since we do not know which send interval between two successive packets drives the routing protocol in a saturated state, we considered the send interval between two successive packet transmissions as the varying parameter, i.e., the interval between two successive packets injected into the network. Clearly, if this interval is too short, the source node is not able to handle all incoming packets, leading to many message drops already in the source node. On the other hand, when the interval is too large, the routing protocol will idle between two successive packets, not exploiting the full available bandwidth for



(a) Average number of hops. **(b)** Data rate in bytes per second.

Fig. 3. The average number of hops and data rate for increasing network size

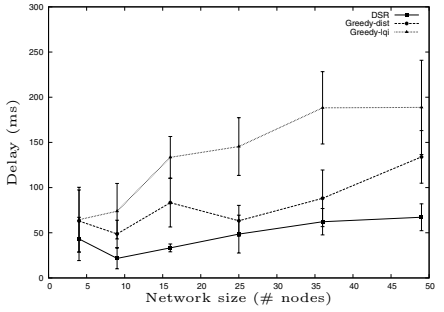
massage forwarding. An optimal saturated state is to be expected in between these two extreme cases. This is confirmed by the measurement results depicted in Fig. 3b. In those measurements we considered the protocols with link layer acknowledgements and drove them with different send intervals. We observed for all protocols a maximum data rate for a send interval ranging from 50 msec to 70 msec.

The maximum data rate achieved both by greedy-lqi and Greedy-dist is close to 500 bytes per second. Maximum data rate achieved by DSR deserves a closer look. DSR requires the path to be stored in the packet. Thus for fixed packet sizes the achievable data rate is correlated with the portion of the packet reserved for the path. In the extreme case no payload will be available, resulting in a data rate of 0. We measured three different variants. *DSR-20* is a conservative implementation, which reserves a fixed size packet header which can store at most 20 hops. *DSR-mhc* (mhc stands for maximum hop count) has also a fixed header size which can store a maximum number of hops. However, this is set to the maximum expected hop count in the scenario. This value has to be determined in advance. Here, we considered the maximum achieved hop count of previous DSR runs in the current experimental setup. Finally, *DSR-var* is a DSR variant which adapts the header length to the actual traversed path. Thus, short paths will automatically provide more payload and thus positively influence the data rate.

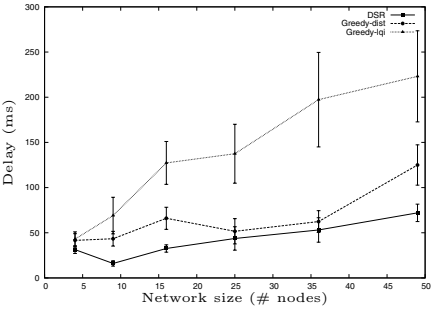
Clearly, setting a conservative fixed header length degrades DSR performance significantly. In the considered example, the maximum performance of DSR-20 drops by factor 3 from the maximum data rate achieved by DSR-var and DSR-mhc. Interestingly, DSR-mhc and DSR-var achieved almost the same performance in our experimental setting. This is an indication, that there are no significant outliers regarding the maximum hop count obtained by DSR. In other words, the maximum hop count is close to the average number of hops by DSR.

For the saturated case, why is DSR performing better than Greedy-dist and why is Greedy-lqi performing less good than DSR and Greedy-dist? We explain the first part of the question by the fact that Greedy-dist is forced to use links to the node closest to the destination notwithstanding the link quality. With DSR in contrast links with bad quality may render RREQ messages getting lost with a high probability. Thus, although DSR and Greedy-dist produce very similar hop counts, DSR is to be expected to use paths which do not contain weakest links. Due to retransmissions sending a message over a weak link requires more time and thus occupies the channel for a longer time. The whole network capacity will be reduced. The same effect can be observed for Greedy-lqi, explaining the second part of our question. While Greedy-lqi selects only high quality links, the number of hops performed by Greedy-lqi is increased. Since we are using fixed signal strengths, i.e., transmission of DSR and Greedy-lqi have the same interference range, the algorithm with less hops per message, DSR, will support higher network capacity.

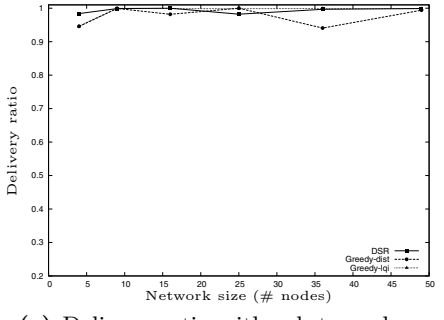
An interesting observation we made in this experiment as well is, when send intervals are increased, superiority of DSR-var and DSR-mhc are interchanged by Greedy-lqi and Greedy-dist. In our measurements this happens for a send interval



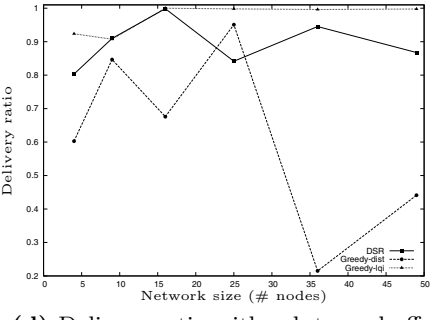
(a) Delay with ack turned on.



(b) Delay with ack turned off.



(c) Delivery ratio with ack turned on.



(d) Delivery ratio with ack turned off.

Fig. 4. Average delay and delivery ratio under increasing network size

of 70 msec. For intervals at or beyond 70 msec both, Greedy-lqi and Greedy-dist, achieve a slightly better performance than DSR. A reasonable explanation for this behavior is missing so far.

Average Delay and Delivery Ratio. Fig.4 depicts the delay and delivery ratio we observed for the protocols, realized with and without link layer acknowledgements. Each measure point is the average over 5 independent routing tasks, while each routing task was sending 500 messages from source to destination. The send interval between two messages was set to 500 msec.

For the delay, depicted in Fig. 4a and 4b, we observed no significant difference when running the protocols with and without link layer acknowledgements. In both cases Greedy-lqi introduced the highest and DSR the lowest delay. The significantly higher delay from Greedy-lqi is reasonable from the previously discussed path diagrams, which suggested that Greedy-lqi favors short edges. The paths taken are thus longer in general. Performance of Greedy-dist is lying in between the performance of DSR and Greedy-lqi. It is closer to the performance of DSR than the one of Greedy-lqi. Clearly, since the hop count distance increases with increasing field size, for all protocols an increase in delay can be observed.

Interestingly, for DSR and Greedy-dist our measurements show a larger latency in the 2×2 network compared with the 3×3 network. Our log files show

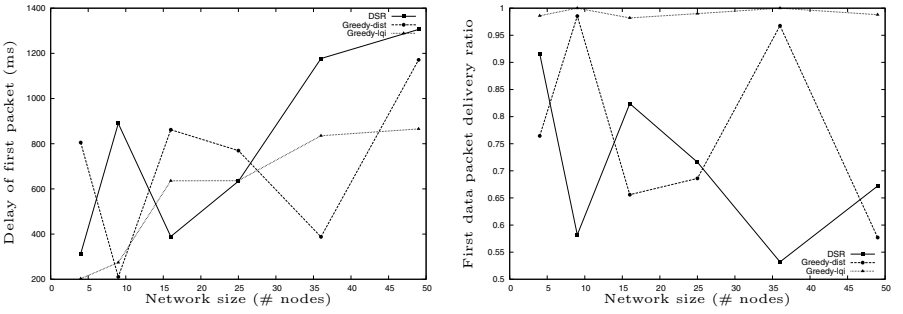
that by coincidence placement and selection of nodes resulted in a 2×2 network where the direct link between source and destination was missing. In contrast, although being longer for the 3×3 network a link between source and destination was available in that network. Although the figures are a result from the average of several experimental runs, the 2×2 and 3×3 networks have only been set up once. We see a possible improvement for future measurements here: Before starting the next experimental run in the same network setting, collect the previous used nodes, mix them with all available nodes, and set up the same network again by randomly selecting nodes form the set of all nodes. This significant amount of additional administrative overhead may pay off with less correlated results from subsequent measurements for one network setting.

We also want to note that our intention here is to depict the average time it takes to get a sequence of message form source to destination. We depict the delay for DSR including the route setup time for the first message. When looking at a single message the delay incurred by route setup and sending the first message completely from source to destination is much higher. For this case our measurements showed a delay ranging from 127 ms for the 2×2 field to 311 ms for the 7×7 field.

As depicted in Fig. 4c, when link layer acknowledgements are turned on, all protocols show a comparable behavior in terms of delivery ratio. All protocols have a delivery ratio clearly above 90%. While for increasing network size DSR and Greedy-dist show some slight variation in delivery ratio, Greedy-lqi appears to achieve a stable delivery ratio close to 100%; a fact which can be explained due to the stable connections hop-wise selected by Greedy-lqi.

When link layer acknowledgements are turned off, DSR and Greedy-dist show high fluctuations in delivery ratio, while delivery ratio of Greedy-lqi still remains above 90% and even close to 100% for larger network sizes. DSR still stays above 80% delivery ratio. Greedy-dist even drops below 30%. While this measurement point appears to be a statistical outlier, the graphs however suggest an ordering with respect to delivery ratio, in most cases Greedy-lqi showing a performance better than DSR, and in most cases DSR showing a better performance than Greed-dist. An explanation for this observation is again the fact that Greedy-lqi selects a stable connection in each hop. DSR in contrast selects the first route getting through to the destination. This path by coincidence may contain links which are less stable. Though, very unstable links are unlikely to be discovered in the route discovery process. In contrast Greedy-dist selects the discovered node closest to the destination, i.e., often a next hop node which is far away from the current node. Due to signal attenuation blindly selecting such large links is counterproductive when aiming on a hop-wise high transmission success.

Delay and Delivery Ratio of the First Packet. In Fig. 5 we depict the delay and the success of getting the first message from source to destination. Each measure point is the average over 500 independent routing tasks. For each task we sent messages until the first message from source to destination was delivered successfully. Running protocols with link layer acknowledgements clearly shows a



(a) Delay until first packet success. **(b)** Delivery ratio of the first packet.

Fig. 5. The first packet's delay and delivery ratio under increasing network size

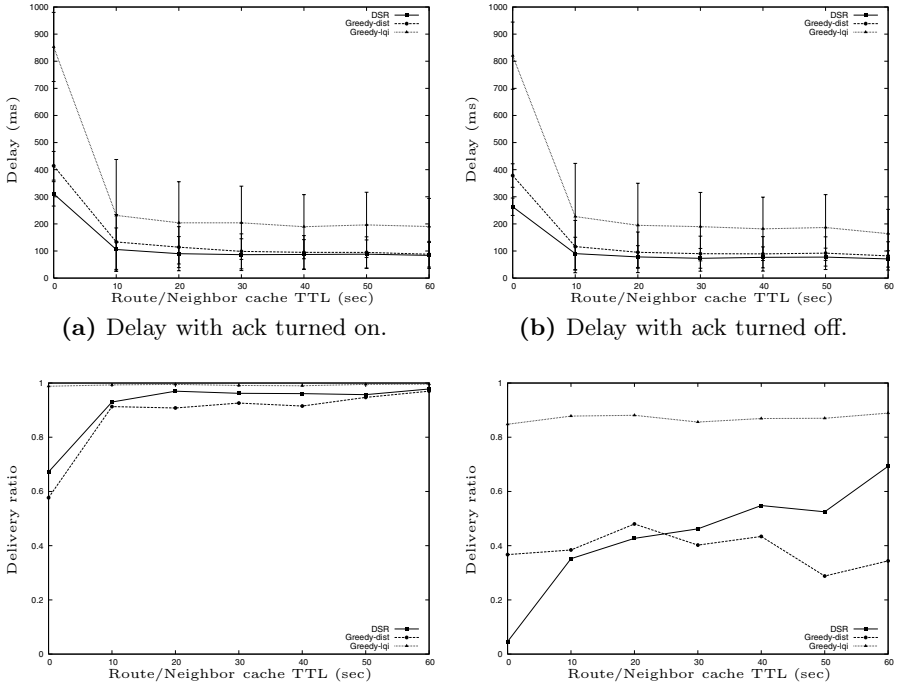
better performance than running them without. Thus, for brevity, in the following we focus on the protocol variants with link layer acknowledgements only.

As can be seen in Fig. 5a, the delay of the first successfully received packet is highly variable for DSR and Greedy-dist. No clear trend can be observed from our measurements. Contrary, Greedy-lqi shows a smooth increase of delay when network size increases. Interestingly, contrary to the average over many messages (see Fig. 4a), when considering first successful reception of a message only, the delay of Greedy-lqi is not clearly above the delay obtained by DSR and Greedy-dist. This is due to the high success of the first Greedy-lqi message as supported by the measurement depicted in Fig. 5b. Greedy-lqi showing a first packet delivery ratio close to 100%, outperformed Greedy-dist and DSR in our measurements. DSR and Greedy-dist are showing the same fluctuations here and no clear trend which one is better can be drawn from our measurements. However, there is an obvious correlation among the DSR and Greedy-dist graphs in Fig. 5a and 5b. A high delivery ratio of the first packet results in a lower delay of the first successful packet reception. Thus, the graph of DSR in Fig 5a appears horizontally mirrored in and 5b. The same applies for Greedy-dist.

4.3 Evaluating a Dynamic Scenario

In addition to the experiments with increasing field size, we investigated the protocol behavior in a dynamic environment. In these measurements we kept the maximum field size of 7×7. Since there is no direct way to add dynamics with reasonable effort to our experimental setup, we "simulate" a dynamic environment by artificially invalidating routes used by DSR and neighborhood tables used by Greedy routing in regular time intervals. By this means we emulate the effect of a dynamic environment where established routes or established neighborhood information frequently get invalid due to topology changes. In DSR, once a path is invalidated, DSR has to set up a new route. In Greedy, once a neighbor list is invalidated, a new neighbor list has to be established.

Fig. 6 depicts the delay and delivery ratio we observed for the protocols, again investigated with and without link layer acknowledgements. Each measurement



(a) Delay with ack turned on.

(b) Delay with ack turned off.

(c) Delivery ratio with ack turned on.

(d) Delivery ratio with ack turned off.

Fig. 6. Average delay and delivery ratio under increasing dynamics

point is an average over 10 independent routing tasks, while in each routing task 100 messages have been sent. The send interval between two successive messages was set to 1 sec. The dynamics parameter was varied in decimal steps from 0 to 60 sec route or neighbor list invalidation time, respectively.

For the delay, depicted in Fig. 6a and 6b we observed no significant difference in protocol performance when link layer acknowledgements are turned on or off. The measurements show the same behavior as observed for the static case. Greedy-lqi introduces the highest and DSR the lowest delay. The performance of Greedy-dist is lying in between, being closer to the performance of DSR than to the performance of Greedy-dist. The reason behind is again explained by the fact that Greedy-lqi is taking significantly longer paths. Clearly, for all protocols delay increases when the dynamics is increased. In the extreme case with an invalidation timeout of 0 sec, delay for all protocols is at maximum. DSR is required to set up a new path for each new message. Both Greedy routing variants are required to rediscover the neighborhood list in each forwarding step.

Finally, Fig. 6c and 6d depicts the delivery ratio when running the protocols with and without link layer acknowledgements, respectively. In both cases Greedy-lqi outperforms delivery ratio of DSR and Greedy-dist, being close to 100% when using acknowledgements and being close to 90% when not using them. In the case of acknowledgements performance of DSR stays above Greedy-dist for all

investigates dynamic values. When not using acknowledgements, interestingly, the roles of DSR and Greedy-dist with respect to delivery ratio are changing. While lower dynamics is better supported by DSR, Greedy-dist becomes better when the invalidation timeout drops below 30 sec. We explain this by the fact that we observed in the log files frequent message losses in the beginning when DSR without link layer acknowledges is setting up a new route. Thus, if route invalidation timeout is increasing, DSR will perform better, approaching its performance for the static case. As discussed in the previous section in the static case DSR shows a better delivery rate compared to Greedy-dist.

5 Conclusion

5.1 Our Present Findings

In this work we presented an empirical performance comparison between a localized and a global routing approach. Although, this work is not the first one providing measurements in a real-world sensor network deployment, we think that in answering the question on what are the advantages and disadvantages of localized over globalized approaches, such empirical results are still lacking.

A summary of the empirical performance comparison in this work is given in the following table. It shows the preferred protocol choice for the according scenarios used in this work. Here, *any* refers to any of the protocols DSR, Greedy-dist, or greedy-lqi, and *Greedy* refers to any of the protocols Greedy-dist or Greedy-lqi. Those possible parameter/scenario combinations we have not investigated in this work are marked with a "-".

Table 1. Preferred protocol choice for the experimental settings

	Static		**Dynamic**	
	ack	no ack	ack	no ack
Delay	DSR	DSR	DSR	DSR
Delivery ratio	any	Greedy-lqi	Greedy-lqi	Greedy-lqi
First contact delay	any	-	-	-
First packet delivery ratio	Greedy-lqi	-	-	-
Data rate at full speed	DSR	-	-	-
Data rate at low speed	Greedy	-	-	-

A general conclusion from this table: When delay of successful packets has highest priority, DSR is the preferred choice. In a static scenario with the use of acknowledgements all protocols achieve a good delivery ratio. In harsh environments, when no acknowledgements are available, or if available but the network gets dynamic, Greedy-lqi shows its benefits resulting from the on demand high quality link selection. In scenarios where getting a single small piece of data to a receiver is of importance, the protocols are not clearly distinguishable with respect to delay. In terms of delivery ratio, however, Greedy-lqi is the preferred choice. Finally, when driving the protocols in a saturated state, DSR performs best in terms of data rate. If packets are injected below protocol saturation, the Greedy variants perform better.

5.2 Open Research Directions

Our intention is to substantiate advantages and disadvantages of localized over centralized data communication protocols with real-world measurements. We focused on one representative of localized and one representative of global routing so far. The space of possible future investigations is endless and we anticipate following the most interesting ones in our future research.

The protocols themselves enable different implementation variants. For instance, in Greedy routing other localized forwarding metrics might be considered and compared to their global counter parts. Moreover, comparison against DSR using link quality measures as well is a possible next step of investigations. Most important, other opponents in the domain of global routing protocols might be considered as well. Finally, for sensor networks one main future track should be the data collecting trees. Here again, localized solutions should be compared against non-localized ones in real-world experimental settings.

The traffic pattern used in this work placed only a light load on the network. We considered just one source communicating to one destination node. Other experiments could contain other traffic patterns stressing the network more than our current setting.

Localized communication protocols require location information. Here we assumed the location information to be configured in advance. This is a reasonable assumption in case of a planned sensor network; which is in most of the practical studies the case. Future performance studies might however incorporate the additional overhead and location inaccuracies of self organized localization techniques; jet opening a new investigation dimension.

There is one important aspect which is still missing in our practical evaluations: energy efficiency. Measuring a protocol's energy consumption in a testbed deployment requires sensor nodes to be attached to additional hardware units which sample the sensor node's current draw during protocol execution. We are aware of two different research groups which have already built such measurement units [13,14]. The described prototypes are so far not commercially available. Resources for reverse engineering or developing a new prototype were not available at the time. We focused our investigations on traditional networking parameters. For future experimental studies we have plans to build energy measurement hardware which allows us to backtrack the sources of energy dissipation in testbed studies.

Finally, we think that a final justification when a localized is better than a globalized approach requires many independent measurements coming from independent research groups. Even measurements repeating experimental setups and experiments of certain protocols are of importance here. We advertise that a single result of one paper should not exclude other papers performing the same measurements and probably achieving varying results. This is however often not that accepted in network research like it is accepted in other disciplines like physics, for instance.

References

1. Royer, E.M., Toh, C.K.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications 6(2), 46–55 (1999)
2. Frey, H., Stojmenovic, I.: Geographic and energy aware routing in sensor networks. In: Stojmenovic, I. (ed.) Handobook on Sensor Networks. Wiley, Chichester (2005)
3. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesc language: A holistic approach to networked embedded systems. In: ACM SIGPLAN Conference on Programming Language Design and Implementation (2003)
4. 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, pp. 93–104 (2000)
5. Buonadonna, P., Hill, J., Culler, D.: Active message communication for tiny networked sensors. In: Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001) (April 2001)
6. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Mobile Computing. Kluwer Academic Publishers, Dordrecht (1996)
7. Takagi, H., Kleinrock, L.: Optimal transmission ranges for randomly distributed packet radio terminals. IEEE Transactions on Communications 32(3), 246–257 (1984)
8. Finn, G.G.: Routing and addressing problems in large metropolitan-scale internetworks. Technical Report ISI/RR-87-180, Information Sciences Institute (ISI) (March 1987)
9. Kuruvila, J., Nayak, A., Stojmenovic, I.: Greedy localized routing for maximizing probability of delivery in wireless ad hoc networks with a realistic physical layer. In: CD Proceedings of the 1st International Workshop on AlgorithmS for Wireless And mobile Networks (A-SWAN), Personal, Sensor, Ad-hoc, and Cellular Workshop (at MobiQuitous), Boston, Massachusetts (August 2004)
10. Heissenbüttel, M., Braun, T.: BLR: Beacon-less routing algorithm for mobile ad-hoc networks. Elsevier's Computer Communications Journal (2003)
11. Blum, B.M., He, T., Son, S., Stankovic, J.A.: IGF: A state-free robust communication protocol for wireless sensor networks. Technical Report CS-2003-11, Department of Computer Science, University of Virginia, April 21 (2003)
12. Füßler, H., Widmer, J., Käsemann, M., Mauve, M., Hartenstein, H.: Contention-based forwarding for mobile ad-hoc networks. Ad Hoc Networks 1(4), 351–369 (2003)
13. Jiang, X., Dutta, P., Culler, D., Stoica, I.: Micro power meter for energy monitoring of wireless sensor networks at scale. In: Proceedings of the 6th International Symposium on Information Processing in Sensor Networks (IPSN 2007), April 25–27, 2007, pp. 186–195 (2007)
14. Köpke, A., Wolisz, A.: Measuring the node energy consumption in USB based WSN testbeds. In: Workshop Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS 2008), June 17–20, 2008, pp. 333–338 (2008)

Multi-hop Cluster Hierarchy Maintenance in Wireless Sensor Networks: A Case for Gossip-Based Protocols

Konrad Iwanicki[1,2] and Maarten van Steen[1]

[1] Vrije Universiteit, Amsterdam, The Netherlands
[2] Development Laboratories (DevLab), Eindhoven, The Netherlands
{iwanicki,steen}@few.vu.nl

Abstract. Multi-hop cluster hierarchy has been presented as an organization for large wireless sensor networks (WSNs) that can provide scalable routing, data aggregation, and querying. In this paper, we revisit the fundamental problem of maintenance of such a hierarchy. To this end, we observe that, due to tightly coupling their operation with the topology of the hierarchy, existing state-of-the-art cluster hierarchy maintenance protocols may not necessarily be efficient. Based on our observations, we make a case for a novel gossip-based hierarchy maintenance protocol that decouples its operation from the hierarchy topology. Through experiments with actual embedded implementations we have developed, we confirm that our protocol can outperform the existing state-of-the-art solutions by a few factors in terms of both the energy consumption and the latency of bootstrapping and recovering the hierarchy.

1 Introduction

Many of the proposed wireless sensor network (WSN) applications assume large node populations deployed over sizable areas. Due to inherent resource limitations of most of wireless sensor node platforms, large multi-hop WSNs must be organized in a way that enables scalable routing, data aggregation, and querying. Moreover, to minimize the effort involved in the deployment and upkeep of a large WSN, it is highly desirable that the organization should be maintainable with minimal human intervention.

Multi-hop cluster hierarchy is a prominent example of such an organization [1,2,3,4]. This hierarchy is a virtual multi-level overlay on the physical network topology: based on their connectivity, nodes are grouped into clusters at level 1, which in turn are grouped into superclusters at level 2, and so on at higher levels. The overlay provides addressing and routing mechanisms, which require only $O(logN)$ node state, thereby offering excellent scalability. These basic mechanisms can be further used to build more advanced services such as distributed hash tables [1,4] or multi-level in-network aggregation and querying [3,5]. Moreover, the overlay can be synthesized and maintained without human intervention, thereby minimizing deployment efforts and facilitating unattended operation. For these reasons, a number of WSN application proposals can be founded on a multi-hop cluster hierarchy, for example, object tracking [1], reactive tasking [6], energy-efficient centralized data collection [3], scalable network monitoring [7], and multi-dimensional range querying [5], to name a few. By and large, multi-hop cluster hierarchy is a compelling paradigm for organizing nodes in large WSNs.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 102–117, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The key component in a system based on a multi-hop cluster hierarchy is the hierarchy maintenance protocol. Such a protocol not only determines the target hierarchy properties and thereby the costs of routing, data collection, and querying, but also generates additional costs due to maintaining these properties during system lifetime. The maintenance costs can easily account for the majority of energy consumed by the nodes. Therefore, since WSNs operate on tight energy budgets, it is essential that the energy costs of hierarchy maintenance are minimized. Even a few-percent improvement in energy consumption can substantially reduce the expenses involved in the upkeep of a large network or can even enable applications that require the prolonged lifetime. Likewise, the hierarchy maintenance protocol should minimize the time to bootstrap the hierarchy and recover it after changes in the network. This affects the availability of the overlay services (e.g., routing, aggregation, querying), especially since large WSNs are less stable in terms of node population and connectivity than their smaller counterparts. The challenge in cluster hierarchy maintenance is the fact that minimizing energy consumption and minimizing bootstrap and recovery time are typically conflicting goals.

In this paper, we revisit the problem of multi-hop cluster hierarchy maintenance in large WSNs with the goal of minimizing the energy consumed to maintain the hierarchy and the time to bootstrap and recover the hierarchy. In contrast to prior work, which focuses on clustering heuristics, we revisit the common general protocol scheme.

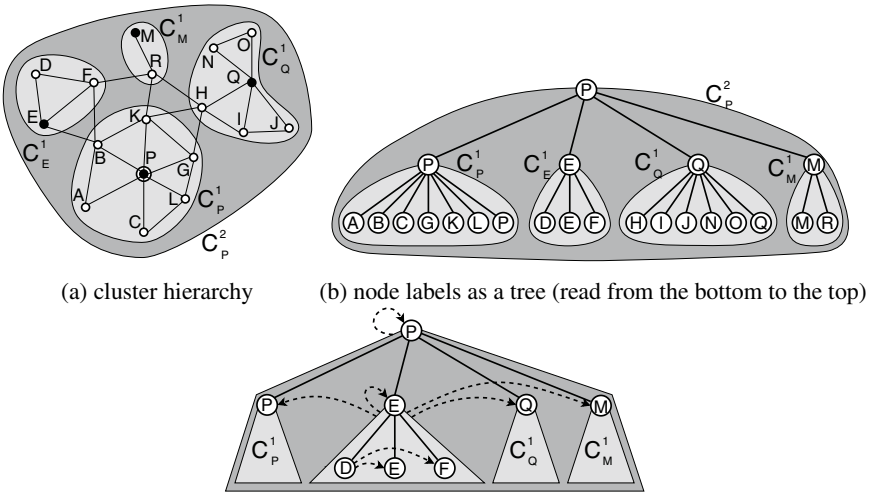
More specifically, we observe that maintaining a hierarchy with the existing state-of-the-art protocols for WSNs is expensive in terms of energy. This is because the operation of all these protocols is *tightly coupled* with the topology of the hierarchy, leading to many small messages being exchanged. In WSNs, however, small messages are typically inefficient, as they incur significant energy overhead on the actual protocol data.

Based on this observation, we propose an alternative protocol that, to improve the performance of hierarchy maintenance, *decouples* its operation from the topology of the hierarchy. The protocol employs a combination of local operations for updating the hierarchy and periodic local gossiping (i.e., asynchronous state exchanges between neighboring nodes) for propagating updates and advertising clusters. This results in more efficient traffic: irrespective of the hierarchy topology, each node periodically transmits only a single big message. Through experiments with actual embedded implementations, we show that our protocol can outperform the existing state-of-the-art protocols, while providing the same hierarchy properties.

The rest of the paper is organized as follows. We formulate the problem and survey prior work in Sect. 2. We explain our protocol in Sect. 3 and evaluate it against existing solutions in Sect. 4. Finally, in Sect. 5, we conclude. Due to space constraints, the code listings and supporting proofs have been moved to a technical report [8].

2 Background and Related Work

Since multi-hop cluster hierarchies have many potential applications in WSNs and other environments, they have received significant research attention. Our protocol builds upon prior work by allowing for using most of the existing clustering heuristics, developed and carefully tuned for particular WSN applications or deployment settings. In this way, the protocol can maintain hierarchies with the same properties as in the existing solutions, and thus, it can be used in the same applications. However, our protocol



(a) cluster hierarchy (b) node labels as a tree (read from the bottom to the top)

(c) routing table of node D (label $D.E.P$)

Fig. 1. An example of a multi-hop cluster hierarchy

follows a novel design paradigm that results in a completely different general operation scheme as compared to the existing protocols. As we show in this paper, this new scheme can improve the efficiency of hierarchy maintenance by significantly reducing the energy consumption and the hierarchy bootstrap and recovery latencies.

In the remainder of this section, we first formulate the problem and then survey prior work. For illustration purposes, we assume a common set of target cluster hierarchy properties [1, 2, 3, 4]. However, as mentioned above and signaled in the protocol description, one can easily obtain different hierarchy properties by simple modifications to the protocol and by employing different clustering heuristics.

2.1 Problem Formulation

A multi-hop cluster hierarchy (see Fig. 1a) consists of multiple levels, on the order of $O(logN)$ with respect to the number of nodes. A node belongs to exactly one cluster at each level, with level-0 singleton clusters that correspond to individual nodes and one or a few top-level clusters that contain all nodes. Each cluster has a *cluster head* that maintains the cluster and can have other roles in some applications, for instance, as an aggregator node for the cluster. Assuming that each node has a unique identifier, a cluster, C^i_X, is uniquely identified by its level, i, and the identifier of its head, X.

The cluster hierarchy is reflected in the *labels* of the nodes. A node's label is a concatenation of the cluster head identifiers for all the clusters the node is member of (see Fig. 1b). For instance, the label of node D from Fig. 1, which is a level-0 cluster head, is $L(D) = D.E.P$ as D belongs to clusters: C^0_D, C^1_E, and C^2_P. The label of node E, a level-1 cluster head, is $L(E) = E.E.P$ as E belongs to clusters: C^0_E, C^1_E, and C^2_P. Finally, the label of node P, a level-2 cluster head, is $L(P) = P.P.P$ as P belongs to clusters: C^0_P, C^1_P, and C^2_P.

Based on its label, each node also keeps a $O(logN)$ *hierarchical routing table* [9]. At each level, a node's routing table contains entries for the heads of sibling clusters at this level (see Fig. 1c). With node labels acting as routing addresses, the routing tables of all nodes enable hierarchical point-to-point routing [1,9]. Since routing is not used by our protocol but only by the applications on top, we omit the algorithm for brevity.

The *cluster hierarchy maintenance protocol*, therefore, is responsible for *maintaining the labels and the routing tables of all the nodes throughout the system's lifetime*. The longevity of a WSN system and its interactions with the physical environment imply changes in the node population and connectivity over time. Hence, to account for the changes, the hierarchy maintenance must be performed continuously rather than just upon system deployment. Such continuous maintenance must *consume minimal amounts of energy* and *provide fast hierarchy bootstrap and recovery after changes*.

2.2 Prior Work

In earlier work on hierarchical routing [9], Hagouel proved that constructing an optimal cluster hierarchy is an NP-complete problem. Thus, only heuristic solutions are practical. Many such heuristics have been developed in data mining for partitioning data sets with respect to a given parameter [10]. Those algorithms, however, require centralized control and fail to scale to large networks of resource-constrained devices.

For this reason, distributed protocols have been introduced. One family of such protocols provides single-level (flat) clustering [11,12,13]. While flat clustering is important in WSN applications using local sensor collaboration for event detection, the definition of cluster hierarchy implies multiple levels. Yet, flat clustering algorithms cannot be easily generalized to efficiently support a multi-hop cluster hierarchy, as their traffic pattern precludes multiple levels and long multi-hop internode distances.

Another family of clustering protocols aims at reducing the energy cost of centralized data collection in dense WSNs [14,15,16]. To this end, the nodes maintain a cluster hierarchy by dynamically adjusting their transmission power and thereby communication range, such that on average the energy consumed by data transmission is minimal. All these algorithms assume that by increasing transmission power, each node can directly communicate with any other node (i.e., one-hop communication). However, while this assumption may hold in WSNs deployed densely in small areas, for practical reasons, it does not hold hold in large-scale real-world WSNs.

These limitations led to a family of cluster hierarchy maintenance protocols aimed specifically at large multi-hop networks of low-power wireless devices [1,2,3,4]. In these protocols, nodes self-organize into a recursive hierarchy of clusters by grouping connected nodes into clusters, grouping clusters into superclusters, and so on. Such a bottom-up approach is better suited for WSNs than an alternative top-down approach [17], which has difficulties with varying deployment parameters, like node densities [1].

Although these state-of-the-art bottom-up protocols vary in clustering heuristics, they follow the same general scheme. In this scheme, nodes advertise their clusters periodically, so that other nodes join such clusters or probabilistically spawn their own higher-level clusters, thereby constructing the hierarchy. The whole process of cluster advertising and update propagation is founded on *hierarchical beaconing*, which is a multi-hop adaptation of hierarchical clustering for dense one-hop networks [14,15,16].

In hierarchical beaconing, a level-i cluster head periodically advertises its cluster by issuing a beacon message that is flooded over R_i hops. To ensure $O(logN)$ hierarchy levels, R_i depends exponentially on i (typically $R_i = 2^i$). Issuing a level-i beacon is expensive, as all nodes within R_i hops from the cluster head must forward the beacon. Therefore, the protocols usually amortize the costs of higher-level beacon forwarding over time by making the number of periods (rounds) between subsequent beacons proportional to the beacon propagation radius. For example, a level-0 cluster head issues a R_0-hop beacon every R_0 rounds, a level-1 cluster head — a R_1-hop beacon every R_1 rounds, and so on. This reduces the beacon forwarding cost at the expense of an increase in the hierarchy bootstrap and recovery latency. To mitigate this increase, a cluster head also issues a beacon whenever it modifies its label, to propagate the update fast.

3 Gossip-Based Hierarchy Maintenance

While these state-of-the-art protocols are elegant, their efficiency, crucial for most applications, can be improved considerably. To conserve energy on idle listening, sensor nodes power their radios off when idle. Such radio activity scheduling, however, incurs significant energy overhead on message transmission or reception, as the sender and receivers must coordinate to have their radios on during the transmission. Very often this overhead outweighs the cost of actual data transmission. For instance, the standard TinyOS MAC layer for WSNs [18] precedes a message with a 100- to 2000-ms preamble, which is a lot compared to less than 0.4 ms necessary to transmit a 12-byte beacon payload (in our implementation). Energy-efficient protocols should thus minimize the overhead on the exchanged data by using fewer but longer messages [18, 19].

However, this is essentially not happening for hierarchical beaconing, which generates myriads of small messages. In preliminary experiments for a hierarchical cluster-based system we are building, for example, to ensure reasonable data availability in the presence of failures, beacon messages could account for more than 90% of energy spent on communication. This also illustrates that minimizing energy overhead incurred by the hierarchy maintenance protocol is crucial for any hierarchical cluster-based system.

3.1 Principal Idea

We observe that the protocols using periodic hierarchical beaconing are so expensive because hierarchical beaconing is *tightly coupled* with the cluster hierarchy. Every beacon is dedicated for one cluster and at every level, i, each node forwards beacons of all level-i heads that are within R_i hops.

Consequently, to reduce the cost of cluster hierarchy maintenance, we propose a protocol that *decouples* its operation from the topology of the hierarchy. Our protocol is based on a combination of local-only operations for updating the hierarchy and periodic local gossiping (i.e., asynchronous state exchanges between neighboring nodes) for propagating such updates and advertising clusters.

The protocol operates in rounds[1]. Once per round, each node broadcasts its protocol state in a single heartbeat message. The state consists of the node's label, update vector

[1] The rounds are local for each node, that is, the node clocks do not have to be synchronized.

(Sect. 3.2), and routing table entries. The heartbeat message is local: it is received only by the node's neighbors and is not forwarded by them. Likewise, in every round, these neighbors broadcast their own state in their heartbeats. The received neighbor state is merged with the node's own local state, so that the node can learn about any changes that have recently occurred in the hierarchy. At the end of its round, the node checks its local state to detect any such changes and to account for them by modifying its state locally. The modified state is broadcast in the node's heartbeat in the next round.

Hierarchy information is thus propagated implicitly, by repeatedly merging the node's state with the state received from its neighbors and broadcasting such a merged state in the next heartbeat message. This is in contrast to periodic hierarchical beaconing, in which the information is propagated explicitly, by forwarding multiple beacon messages dedicated for specific clusters. The result of this paradigm shift is smaller energy overhead on protocol data: instead of forwarding multiple small beacons per round, each node transmits only a single big heartbeat message.

The paradigm shift, however, requires revisiting the mechanisms for maintaining the hierarchy, which is our focus in the rest of this section. Since formally describing the protocol is outside the scope of this paper, we aim only at explaining it adequately. The pseudo-code and proofs, in turn, can be found in our technical report [8].

3.2 Update Vector

One of the key revisited mechanisms is hierarchy membership update resolution. Since in hierarchical beaconing every member of a cluster receives beacons issued by the head of this cluster, it has direct access to the label of the head. Therefore, when the head modifies its label and issues a beacon, the cluster members can consistently apply the updates to their labels. In contrast, with local gossiping, a node has access only to the labels of its immediate neighbors, and there is no way to determine which of the neighbors' labels contain fresher membership information. For example, without additional information, a node with label $A.K.\mathrm{P}.R.S$, receiving from its neighbors heartbeats with labels $B.K.\mathrm{P}.Q.T$ and $C.J.\mathrm{P}.U.V$, cannot determine which of the three labels contains the freshest information on the membership of cluster C_P^2 in the hierarchy.

To solve this problem, we introduce *update vectors*. A node's update vector corresponds to the node's label and unambiguously specifies the updates applied to the label. The i-th element of the vector denotes the sequence number of the last known label update made at level $i{+}1$ by the node's level-i cluster head, as illustrated in Fig. 2. A node's

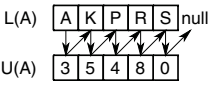


Fig. 2. A label and update vector. *A knows that: (i) the last level-1 label update of A has number 3 and wrote K at position 1; (ii) the last level-2 update of K has number 5 and wrote P at position 2; (iii) the last level-3 update of P has number 4 and wrote R at position 3; (iv) the last level-4 update of R has number 8 and wrote S at position 4; (v) S has not yet made any level-5 updates ($U(A)[4] = 0$).*

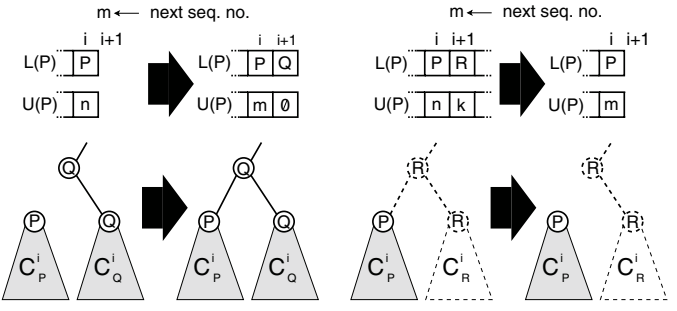


Fig. 3. Label extension and label cut

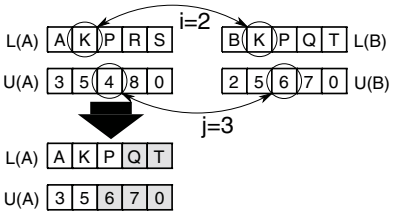


Fig. 4. Label resolution and update adoption

update vector is broadcast with the node's label in the node's heartbeat messages and is essential to synthesizing and maintaining node labels.

3.3 Basic Label Operations

To maintain their labels, the nodes use three basic operations: label extension, label cut, and label resolution. Label extension (see Fig. 3a) is executed locally by a top-level cluster head, P, during hierarchy construction or recovery. It corresponds to joining cluster C_P^i to a higher-level cluster, C_Q^{i+1} (if P's label is extended with Q), or spawning a new higher-level cluster, C_P^{i+1} (if P's label is extended with P). Label cut (see Fig. 3b), in turn, is executed locally by a non-top-level head, P, during hierarchy recovery when P detects that its supercluster, C_R^{i+1}, no longer exists as, for example, its head, R, may have died. This operation corresponds to removing cluster C_P^i from the no longer existing higher-level cluster, C_R^{i+1}. In both operations, when modifying its label at level i+1, node P also writes a new sequence number at the i-th position of its update vector. This is to indicate that this label update by P is the freshest one, so that other members of C_P^i can also adopt the update through label resolution.

Label resolution is thus the way to propagate label updates in our protocol. It is done every time a node, A, receives a heartbeat from a neighbor, B, and works as follows (see Fig. 4). A checks if it shares a cluster with B, that is, A looks for the minimal i such that $L(A)[i] = L(B)[i]$. If such i exists, A checks which of the labels is fresher by comparing its update vector, $U(A)$, with B's update vector, $U(B)$, from position i. If for some $j \geq i$, $U(A)[j] \neq U(B)[j]$ then one of the labels is fresher than the other

(they can differ from the j+1-st element). If B's label is fresher ($U(A)[j] < U(B)[j]$), then A copies B's label and update vector from position $j+1$ and j, respectively: $L(A)[j+1 \ldots \mathcal{H}] \leftarrow L(B)[j+1 \ldots \mathcal{H}]$ and $U(A)[j \ldots \mathcal{H}] \leftarrow U(B)[j \ldots \mathcal{H}]$. In this way, A's hierarchy membership information becomes consistent with the fresher information from B. Moreover, when A broadcasts its next heartbeat, its other neighbors also adopt the fresh information, and so on. In our technical report [8], we formally prove that this algorithm guarantees consistent label update adoption by all cluster members.

3.4 Maintaining Routing Tables

While label resolution is performed for every received heartbeat, the usage of label extension and cut is dependent on the state of a node's routing table. Node routing tables are maintained with a custom distance-vector algorithm.

A routing entry corresponds to a cluster (cf. Sect. 2.1). It consists of the level and the identifier of the cluster head, a sequence number, the link-layer address of a next-hop neighbor on the path to the head, the number of hops on this path, and an additional bit indicating whether the cluster is adjacent to the present node's cluster at the same level.

An entry for a cluster originates at the cluster head, which sets the hop count of the entry to zero and generates a new sequence number for the entry. Generating the sequence numbers follows the same pattern as issuing beacon messages in hierarchical beaconing: a level-i head generates a new sequence number for its cluster entry every R_i rounds. Alternatively, a new sequence number can be generated in each round.

Clusters are advertised via heartbeat messages. When the head broadcasts a heartbeat, its neighbors can refresh the entries for the head's cluster with a new sequence number. When they broadcast their heartbeats, their neighbors can also refresh their routing entries, and so on up to R_i hops. More specifically, a heartbeat broadcast by a node contains those entries from the node's routing table that were refreshed in the past round (or k rounds if we want to tolerate message loss). A node receiving a heartbeat refreshes those entries in its routing table that are present in the heartbeat and have fresher sequence numbers than the node's own entries. In addition, since the same routing entry can be present in heartbeats of different neighbors, the node must choose one of the neighbors as the next hop for the routing entry. As in the distance-vector algorithm, it chooses the neighbor whose routing entry has had the smallest hop count.

When a node does not refresh a routing entry for a certain number of rounds, depending on R_i, it concludes that the cluster corresponding to the entry is no longer reachable, for instance, because the head died. Consequently, the entry can be removed from the node's routing table. To prevent routing cycles in the presence of node failures, we use route poisoning: before removing an entry a node marks it as unreachable and broadcasts such an entry in its heartbeat messages for several rounds, thereby allowing other nodes to learn about the failure as well.

For a given cluster hierarchy, the routes maintained by the above algorithm are the same as the routes maintained by periodic hierarchical beaconing. This is because the *rules* for constructing the routes are the same. However, the protocols employ completely different *paradigms* for propagating route information. In periodic hierarchical beaconing, every routing entry of a node is refreshed by a separate beacon message received by the node. In our protocol, in contrast, a single heartbeat message can refresh

multiple routing entries of the receiving node. As a result, our protocol maintains the same routing tables more efficiently as we demonstrate empirically in the next section.

3.5 Synthesizing and Maintaining Labels

Based on its routing table, each node synthesizes and maintains its label with the three basic label operations. During a round, a node receives heartbeat messages from its neighbors. The heartbeats are used for refreshing the node's routing table and for resolving and adopting any label updates made by the heads of the node's clusters. At the end of the round, the node, being itself a cluster head at some level, analyzes its routing table and, as a result, possibly extends or cuts its label locally to account for any network changes that have occurred. The possibly updated node label is broadcast in the node's heartbeat in the next round, allowing the members of the node's cluster to gradually adopt the label updates. By repeating this scheme in every round, the nodes synthesize and continuously maintain their labels.

Label Synthesis. Initially, each node is a top-level head of its level-0 cluster, as its label contains only one element. If the node discovers in its routing table an entry for another cluster head at the same or a higher level, it has to either spawn a new higher-level cluster itself or join the higher-level cluster of the other node. In the first case, it would extend its label with its own identifier, promoting itself to a higher-level head. In the second case, it would extend its label with the identifier of the other node (see Fig. 3a).

Joining an existing cluster is preferred, as it decreases the number of clusters at subsequent levels. However, depending on clustering heuristics, joining may not always be possible. In that case, our protocol utilizes the same heuristics as the existing state-of-the-art protocols [1,2,3,4]. More specifically, the node probabilistically defers spawning the higher-level cluster by drawing a random promotion slot and waiting for this slot. If during this time other nodes in a similar situation spawn new higher-level clusters, the node may be able to join one of such clusters. Otherwise, the node spawns its own higher-level cluster. In any case, when the node broadcasts its heartbeat after extending its label, the node's neighbors that belong to the node's cluster can also extend their labels through the label resolution operation. When they broadcast their heartbeats, their neighbors that belong to the node's cluster can extend their labels as well and so on. In our technical report, we prove that this ensures probabilistic label convergence [8].

Label Recovery. When a node has died, it no longer advertises its cluster. Thus, other nodes do not refresh the routing entries for that cluster. If a node has not refreshed a routing entry for a certain number of rounds, the entry is evicted from the node's routing table. If a node, being a level-i cluster head, discovers that the entry for its parent level-$i{+}1$ head has been evicted, it concludes that its cluster must not be a subcluster of the no longer existing level-$i{+}1$ cluster. Hence, it cuts its label down to level i (see Fig. 3b). Later, by virtue of the above label synthesizing mechanisms, the dangling cluster of this node will join to some other higher-level cluster, completing the recovery.

By employing the same probabilistic clustering heuristics as the existing state-of-the-art protocols, our solution constructs the same cluster hierarchies as those protocols. One of the consequences of building upon these well-established heuristics is that our protocol can be utilized in the same applications, for instance, to improve performance.

Table 1. Memory breakdown of the test application for a TelosB node. *Both protocols require little memory, with their RAM footprints being dominated by the pools for the routing table entries. Due to the more sophisticated operation scheme, however, our protocol (Decoupled) requires slightly more code and data space. The total data space, in turn, is higher for the application with the other protocol (Coupled), as it requires more message buffers (for beacon messages).*

Variant / Component	Coupled		Decoupled	
	RAM	ROM	RAM	ROM
Protocol Only	1,382 B	3,600 B	1,469 B	4,269 B
Total	4,408 B	22,452 B	4,094 B	23,774 B

4 Experimental Evaluation

We evaluate the existing state-of-the-art solutions against our approach using actual embedded implementations. To the best of our knowledge, this is the first reported implementation-based evaluation of multi-hop cluster hierarchy maintenance protocols for WSNs. While prototyping our protocol, we also conducted extensive experiments with a custom simulator. Those results are presented in our technical report [8].

4.1 Protocol Implementations

The discussed existing multi-hop cluster hierarchy maintenance protocols for WSNs [1,2,3,4] operate according to the same common scheme, founded on hierarchical beaconing, but differ slightly in clustering heuristics. Therefore, we have implemented the most recent protocol with arguably the most efficient heuristics [3]. To enable fair comparison, we have used the same heuristics in the implementation of our solution. In general, where possible, both implementations use precisely the same components. As a result, both the implementations build hierarchies with the same target properties, and differ only in mechanisms for maintaining these properties. We believe that this is a sound approach to show the performance gains that can be obtained with our protocol.

For the purpose of the evaluation, we have written a simple test application that contains only statistic reporting functionality and either of the protocol implementations. The test application enables testing hierarchy maintenance in isolation from other services present in a complete real system. We leave the evaluation of a complete representative system for future work. In the remainder of this section, we use *Coupled* to refer to the variant of the application with the state-of-the-art protocol based on periodic hierarchical beaconing, and *Decoupled* to refer to the variant with our protocol.

As the implementation platform for the protocols and the test application, we have chosen TinyOS 2.0. To estimate link quality, which is necessary for discriminating node neighbors from poorly connected nodes, we use the standard link estimator based on exponentially weighted moving average of packet reception rate [20]. This estimator is fast and portable, and offers acceptably accurate link estimates. As the MAC layer, we use the standard TinyOS 2.0 MAC, that is, CSMA/CA with low-power listening [18]. This MAC layer is well suited for hierarchical cluster-based systems as it provides low energy consumption, scales to large networks, and efficiently handles varying workloads and network dynamics. The memory breakdown of the test application for each of the hierarchy maintenance protocols is presented in Table 1.

4.2 Experimental Setup

We have conducted our experiments on a small indoor testbed consisting of 55 TelosB nodes [21] and in TOSSIM, a low-level TinyOS simulator. In both environments, we used precisely the same test application, described above. Since the evaluated protocols display most of their scaling properties only in large networks, starting from some hundred nodes [1,2,3,4,8], due to space constraints, in this paper we focus mostly on the TOSSIM experiments and only briefly summarize the small-scale testbed results.

TOSSIM provides a realistic and accurate simulation environment for TinyOS applications. It captures the TinyOS behavior at a low level and offers signal propagation and noise models derived from real deployments. This enables accurate evaluation of the protocols in realistic settings that give good predictions on the real-world protocol behavior, as we have seen in the testbed experiments.

We have conducted TOSSIM experiments in a number of network configurations. To evaluate protocol scalability, we exponentially varied the node population from 64 to 1024 nodes. The nodes were placed on a square grid. By also varying the grid spacing, we obtained different node densities: from 11.27 (sparse) to 46.42 (dense) good-quality neighbors per node on average. Using traces from real-world deployments and TOSSIM tools, for each configuration, we generated a realistic signal propagation model. The resulting connectivity exhibited many irregularities that are common in real-world WSNs. For example, nearby nodes were often not connected and there were many asymmetric links. As a result, the grid node placement was not mirrored by the neighbor relation. This alleviates the problem that due to page constraints on this paper, we omit other tested topologies (i.e., uniform and random).

In all configurations, a node was identified with 10 bits. The top hierarchy level was 5, as this was enough for a top-level cluster to cover the whole network. Consequently, the label size in a beacon or heartbeat message was at most $\lceil (10 \cdot (5+1))/8 \rceil = 8$ bytes. This was also the size of the update vector in a heartbeat message. A single routing table entry in a heartbeat message, in turn, was 4 bytes long.

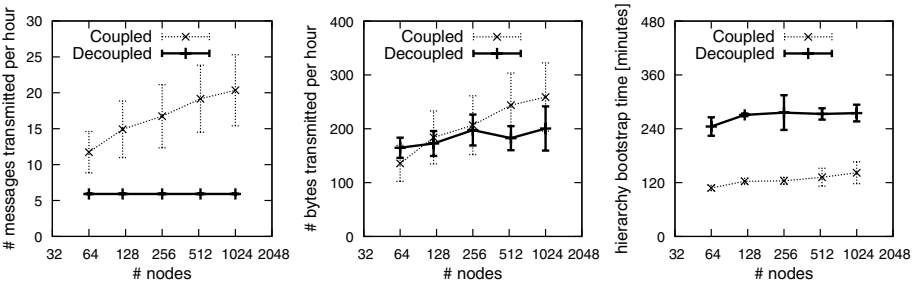
4.3 Experimental Results

When maintaining the desired properties of a multi-hop cluster hierarchy, the goal is to minimize both the energy consumption and the bootstrap/recovery latency. Since these two are often in conflict, our target performance metric is the energy consumption per given time period versus the latency of bootstrapping and recovering the hierarchy.

We start with standard experiments in which all nodes are booted simultaneously and have to construct the hierarchy from scratch. For both protocols, the round length, T, is equal to 10 minutes. This allows us to illustrate the strengths and weaknesses of each of the protocols. Selected results of the experiments are presented in Fig. 5.

As we argued in the previous section, the Coupled protocol uses substantially more messages than our Decoupled protocol (Fig. 5a). In a sparse 1024-node network, for example, it generates more than 20 messages per node per hour (> 3 messages per round), and this value varies significantly between nodes. Our Decoupled protocol, in turn, produces small flat traffic of 6 messages per node per hour (1 message per round).

Likewise, because it advertises clusters less efficiently, the Coupled protocol requires more bandwidth in larger networks (Fig. 5b). Every beacon message has two coupled



(a) messages per node per hour (b) bytes per node per hour (c) bootstrap latency

Fig. 5. Protocol scaling properties. *Each point represents the average and the standard deviation over 10 runs. The round length, T, is* 10 *minutes for both protocols. The networks are sparse. The results for denser networks do not differ significantly. Note also a logarithmic scale of the x-axes.*

objectives: consistently propagating hierarchy membership updates for a cluster and advertising the cluster to refresh node routing tables. To this end, apart from a hop count and a sequence number, a beacon has to store the full label of the cluster head (up to 8 bytes in our implementation). In our Decoupled protocol, in contrast, propagating cluster advertisements is independent of update propagation, which is performed through label resolution. Therefore, instead of a full label, an advertisement of a cluster corresponds only to the routing entry for the cluster (4 bytes in our implementation). Note also that due to maintaining the same target hierarchy properties, both protocols generate roughly the same number of cluster advertisements. Consequently, since in larger networks cluster advertisements dominate the payload of heartbeat messages, our protocol requires less bandwidth, even though it incurs some additional overhead due to transmitting neighbor labels and update vectors. Moreover, due to transmitting less messages, our approach saves bandwidth on TinyOS message headers and footers. These results, however, are not included in the plot.

For the hierarchy bootstrap latency, the relationship is opposite (Fig. 5c). Since a beacon is forwarded by a node shortly after it is received, hierarchy updates in the Coupled protocol propagate fast. In contrast, because heartbeats are broadcast by a node only once per round, update propagation through label resolution is slower. In the most pessimistic scenario, it can take up to d rounds to propagate an update over d hops. For this reason, given the same round length, the Decoupled protocol bootstraps the hierarchy slower than the Coupled one. In the networks plotted, this difference is roughly a factor of two. The bootstrap latencies in the plot seem flat because for implementation purposes we limited the top level to 5. If this is not the case and the bootstrap criterion is a single top-level cluster, the latency grows with the network size [8]. Moreover, the latency can change depending on the number of rounds the link estimator requires to identify good-quality links. In our implementation, this number ranged from 5 to 6.

Since the reduction in energy consumption is typically sublinear with respect to the reduction in the traffic volume, to avoid exaggerating the performance improvements of our protocol, we directly compare the energy consumed by both the protocols. We obtain the energy consumption by following a standard methodology for the underlying MAC layer [18, 19], which produces relatively accurate results. More specifically,

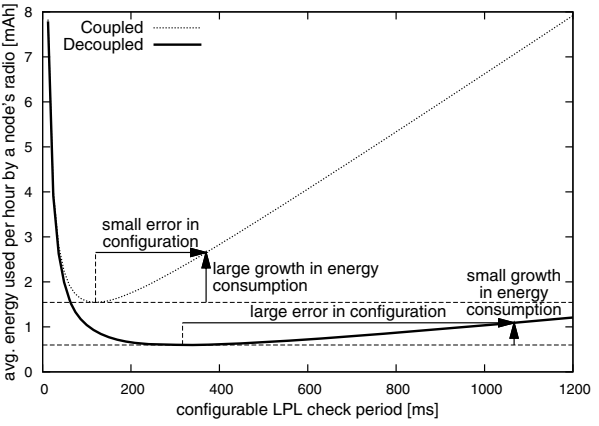


Fig. 6. Energy consumption as a function of the LPL check period. *The round length, T, is* 10 *minutes for both protocols. The network consists of* 1024 *densely deployed nodes. Again, for sparse networks the results are consistent. The current draw measurements correspond to* 250*-kbit/s IEEE 802.15.4 radios (as in TelosBs) and come from Klues et al. [19]. The results for slower radios, such as in Mica2s, are consistent, albeit the difference is slightly smaller.*

we combine radio activity traces from the above experiments with publicly available measurements of the current drawn by a node's radio in various sensor node platforms. Sample results for the TelosB platform are depicted in Fig. 6.

To conserve radio energy, the standard MAC layer for TinyOS 2.0 employs a technique called low-power listening (LPL) [18]. LPL involves one configurable parameter, the LPL check period, which determines how often the radio is turned on to check for a carrier and how long the message preamble is. This parameter reflects the trade-off between energy consumption when the radios are idle and the energy overhead on data transmission. Figure 6 shows that, for the same round length, our Decoupled protocol outperforms the Coupled one for all reasonable settings of the LPL check period. In particular, with the LPL check period configured optimally for each of the protocols, our protocol requires 2.5 times less energy than the Coupled protocol. Moreover, in the Coupled protocol even a small deviation from the optimal setting results in large growth of the energy consumption. This, however, is undesirable as such deviations are expected in the real world because experience shows that WSN applications often exhibit traffic patterns that are difficult to predict prior to the actual deployments.

In the experiments so far, both protocols operated with the same round length, $T = 10$ minutes. In this configuration, the Coupled protocol bootstrapped the hierarchy faster (cf. Fig. 5c), but consumed more energy than our Decoupled protocol (cf. Fig. 6). Therefore, a systematic comparison requires varying the round length, T, as in Fig. 7.

Our Decoupled protocol consistently outperforms the Coupled one, thereby getting closer to the ideal protocol. For example, when the round length for each of the protocols is configured such that both protocols consume the same amount of energy per hour, our protocol bootstraps the hierarchy $\sim$2.6-3.1 times faster. When the two protocols are configured to bootstrap the hierarchy with the same speed, in turn, our Decoupled protocol consumes only $\sim$0.49-0.67 of the energy consumed by the Coupled

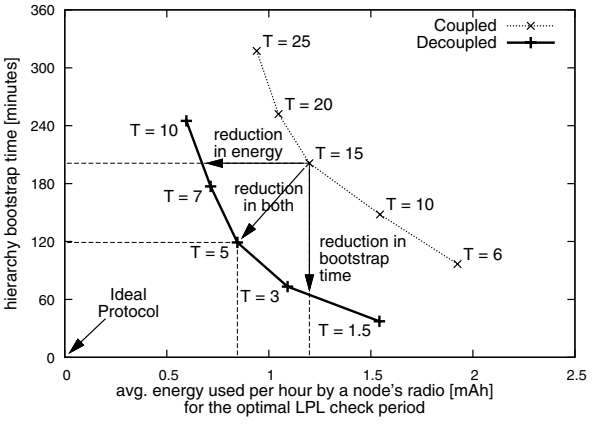


Fig. 7. Energy consumption versus hierarchy bootstrap latency. *Each point represents the average over 10 runs. The network consists of* 1024 *densely deployed IEEE 802.15.4 nodes. Again, the results for different node densities and radios are consistent. The round lengths,* T*, are in minutes.*

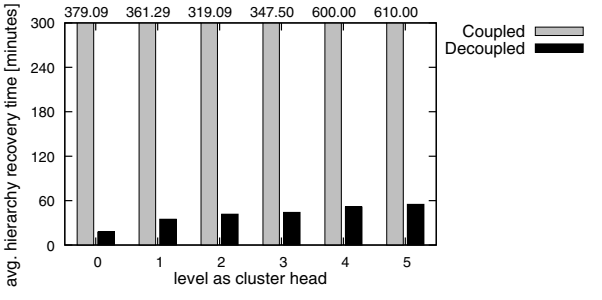


Fig. 8. Latency of recovery after a failure of a cluster head at a given level. *The network consists of* 1024 *sparsely deployed nodes. The round length,* T*, is* 3 *minutes for our Decoupled protocol and* 15 *minutes for the Coupled one as this results in a similar average energy consumption.*

protocol. Moreover, it is possible to configure our protocol to outperform the Coupled one in both the metrics. These results constitute a major performance improvement as 50% reduction in energy consumption typically doubles the network lifetime.

After the above experiments, to compare the performance of failure recovery, we conducted the following micro-benchmarks. We set the round length for both protocols such that they consume roughly the same amount of energy per hour in the stable state. After the hierarchy has converged, we killed a single node and measured the time to recover the hierarchy depending on the node's level as cluster head. By recovery we mean a state in which neither the label nor the routing table of any alive node contains the identifier of the failed node. Reaching this state guarantees that all higher-level services (e.g., routing, aggregation, querying) will operate correctly. Afterward, we reincarnated the dead node and let it fully rejoin the system. We then repeated the above steps for all other nodes in the network. The results of the experiment are presented in Fig. 8.

Table 2. Comparison of the TOSSIM experiments from Fig. 5 with the testbed experiments. *In general, the results match. The small differences in protocol performance stem mainly from the differences in the deployment parameters, like the network size, density, and topology.*

Experimental Setting / Metric Name	TOSSIM		Testbed	
	Coupled	Decoupled	Coupled	Decoupled
Number of Nodes	64		55	
Avg. Number of Neighbors per Node	7.72		19.51	
Avg. Messages per Node per Hour	11.73	6.0	12.44	6.00
Avg. Bytes per Node per Hour	135.75	164.73	143.78	149.30
Hierarchy Bootstrap Time [minutes]	108.02	245.02	135 ± 5	235 ± 5

These results again demonstrate higher efficiency of our protocol. Failure recovery involves two mechanisms: detection and repair. In both protocols, cluster head failure detection requires the same number of rounds, depending on the level of the cluster head. Because our Decoupled protocol performs better than the Coupled protocol when operating with shorter rounds (cf. Fig. 7), it detects cluster head failures faster. Moreover, since the repair is done by the same mechanisms as the hierarchy construction (Sect. 3.5) and since our protocol constructs the hierarchy more efficiently, it is also more efficient when repairing the hierarchy. Consequently, our protocol outperforms the Coupled one also in failure recovery. In the conducted experiment, for instance, our protocol recovered from a level-5 cluster head failure $\sim$11.12 times faster (not visible in the plot), while using the same amounts of energy as the Coupled protocol.

Finally, to verify the TOSSIM results we ran the protocols on our 55-node testbed [21]. Table 2 confirms that the testbed results match the TOSSIM results. Moreover, it indicates that our solution can smoothly operate in the real world.

Altogether, the results demonstrate that by decoupling its operation from the hierarchy topology, our protocol can propagate hierarchy information more efficiently, through less frequent albeit much bigger messages. This allows for shortening the protocol round, and thus, for reducing the bootstrap and recovery latency, but without increasing the energy consumption. While the absolute performance gains may differ in different environments (i.e., hardware platforms, MAC layers, deployment settings), we believe that our protocol has potential to improve the efficiency of large-scale hierarchical cluster-based systems built using wireless low-power devices.

5 Conclusions

We argued that, by proverbially "doing more with less," the efficiency of state-of-the-art multi-hop cluster hierarchy maintenance protocols for large WSNs can be significantly improved. To illustrate our claim, we proposed a novel protocol that maintains the same target hierarchy properties with fewer albeit bigger messages, thereby minimizing energy overhead on the exchanged hierarchy information. These reductions are achieved by having redesigned hierarchy maintenance to use a combination of local updates and periodic local gossiping, which decouples protocol operation from the topology of the hierarchy. As we showed, the reductions in energy consumption and hierarchy bootstrap and recovery latency due to such decoupling can be substantial.

References

1. Kumar, S., Alaettinoglu, C., Estrin, D.: SCalable Object-tracking through Unattended Techniques (SCOUT). In: Proc. IEEE ICNP 2000, Osaka, Japan, pp. 253–262 (2000)
2. Subramanian, L., Katz, R.H.: An architecture for building self-configurable systems. In: Proc. ACM MobiHoc 2000, Boston, MA, USA, pp. 63–73 (2000)
3. Bandyopadhyay, S., Coyle, E.J.: An energy efficient hierarchical clustering algorithm for wireless sensor networks. In: Proc. IEEE INFOCOM 2003, San Francisco, CA, USA (2003)
4. Du, S., Khan, A., PalChaudhuri, S., Post, A., Saha, A.K., Druschel, P., Johnson, D.B., Riedi, R.: Self-organizing hierarchical routing for scalable ad hoc networking. Technical Report TR04-433, Rice University, Houston, TX, USA (2004)
5. Li, X., Kim, Y.J., Govindan, R., Hong, W.: Multi-dimensional range queries in sensor networks. In: Proc. ACM SenSys 2003, Los Angeles, CA, USA, pp. 63–75 (2003)
6. Akyildiz, I.F., Kasimoglu, I.H.: Wireless sensor and actor networks: Research challenges. Ad Hoc Networks 2(4), 351–367 (2004)
7. Iwanicki, K., van Steen, M.: Towards a versatile problem diagnosis infrastructure for large wireless sensor networks. In: Proc. OTM PerSys 2007, Vilamoura, Portugal, pp. 845–855 (2007)
8. Iwanicki, K., van Steen, M.: The PL-Gossip algorithm. Technical Report IR-CS-034, Vrije Universiteit, Amsterdam, the Netherlands (2007)
9. Hagouel, J.: Issues in Routing for Large and Dynamic Networks. PhD thesis, Columbia University (1983)
10. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)
11. Younis, O., Fahmy, S.: Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In: Proc. IEEE INFOCOM 2004, Hong Kong, China, pp. 640–651 (2004)
12. Jia, L., Rajaraman, R., Suel, T.: An efficient distributed algorithm for constructing small dominating sets. Distributed Computing 15(4), 193–205 (2002)
13. Amis, A.D., Prakash, R., Vuong, T.H.P., Huynh, D.T.: Max-min d-cluster formation in wireless ad hoc networks. In: Proc. IEEE INFOCOM 2000, Tel-Aviv, Israel, pp. 32–41 (2000)
14. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocols for wireless microsensor networks. In: Proc. 33rd HICSS, Maui, HI, USA (2000)
15. Manjeshwar, A., Agrawal, D.P.: TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In: Proc. IEEE IPDPS 2001 Workshops, San Francisco, CA (2001)
16. Ye, M., Li, C., Chen, G., Wu, J.: EECS: An energy efficient clustering scheme in wireless sensor networks. In: Proc. IEEE IPCCC 2005, Phoenix, AZ, USA, pp. 535–540 (2005)
17. Thaler, D., Ravishankar, C.V.: Distributed top-down hierarchy construction. In: Proc. IEEE INFOCOM 1998, San Francisco, CA, USA, pp. 693–701 (1998)
18. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: Proc. ACM SenSys 2004, Baltimore, MD, USA, pp. 95–107 (2004)
19. Klues, K., Handziski, V., Lu, C., Wolisz, A., Culler, D., Gay, D., Levis, P.: Integrating concurrency control and energy management in device drivers. In: Proc. ACM SOSP 2007, Stevenson, WA, USA, pp. 251–264 (2007)
20. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proc. ACM SenSys 2003, Los Angeles, CA, USA, pp. 14–27 (2003)
21. Iwanicki, K., Gaba, A., van Steen, M.: KonTest: A wireless sensor network testbed at Vrije Universiteit Amsterdam. Technical Report IR-CS-045, Vrije Universiteit, Amsterdam, the Netherlands (2008)

Potentials of Opportunistic Routing in Energy-Constrained Wireless Sensor Networks

Gunnar Schaefer, François Ingelrest, and Martin Vetterli

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Switzerland
firstname.lastname@epfl.ch

Abstract. The low quality of wireless links leads to perpetual packet losses. While an acknowledgment mechanism is generally used to cope with these losses, multiple retransmissions nevertheless occur. Opportunistic routing limits these retransmissions by taking advantage of the broadcast nature of the wireless channel: sending packets to multiple receivers at once, and only then, based on the outcome, choosing the actual next hop [1]. In this paper, we first study the potentials of opportunistic routing in energy-constrained wireless sensor networks. In particular, the reduction of retransmissions due to the broadcast advantage is balanced with the arising need for coordination to avoid duplicate packets. We then propose Coordinated Anypath Routing, an opportunistic routing protocol designed for wireless sensor networks, in which the coordination between receivers is handled by an overhearing-based acknowledgment scheme. Our protocol may be used to minimize either retransmissions or power consumption, and our simulation results show that, with lossy links, energy savings go up to 7%, even for small networks of 20 nodes.

1 Introduction

Multi-hop routing is a key feature of wireless ad hoc networks. Compared to traditional cellular architectures, ad hoc networks are much more flexible, as they allow users and/or devices to roam freely, without having to worry about access point locations. As long as there exists an unbroken chain of devices from a source to a destination, it is the responsibility of the routing protocol to discover it and to construct a route along it, allowing for long-distance communications.

In wireless sensor networks (WSNs), although nodes are often stationary, multi-hop routing still offers much-desired flexibility in placing, adding, and removing sensor nodes. One such example is SensorScope[1], on which we have worked over the past three years. It is a time-driven WSN, used to gather dense spatio-temporal measures of various environmental parameters. Our most prominent deployment took place on top of the Génépi mountain in the Swiss Alps. The gathered data allowed environmental scientists to detect and model a microclimate, which had been the cause of dangerous mud flows during strong rain

[1] http://sensorscope.ch/

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 118–133, 2009.
© Springer-Verlag Berlin Heidelberg 2009

falls [2,3]. In this particular case, multi-hop routing was a fundamental requirement, because the sink node—equipped with a GPRS transceiver—had to be placed on a nearby ridge to ensure connectivity to the GSM network. As a cellular network architecture would have forced all stations to be close to the sink, it would have prevented us from gathering the desired data.

Although much work has already been published on multi-hop routing for both ad hoc and sensor networks (*e.g.*, OLSR [4], ZRP [5], MintRoute [6]), many papers are still appearing, proposing novel algorithms, each of them trying to improve certain networking aspects (*e.g.*, energy consumption, robustness). All of these protocols, however, generally have one thing in common: at each node, a given packet is forwarded to a single, preselected neighbor. Opportunistic routing takes a different approach: each packet is forwarded to a *set* of neighbors, instead of only one [1,7]. Obviously, the probability that *at least one* node in this set receives a packet is much higher than that of a particular node receiving it. In working on the SensorScope system, we became very interested in this simple, yet intriguing idea, and have worked on adapting it to wireless sensor networks.

In this paper, we present our work on opportunistic routing in data gathering WSNs. Section 2 discusses common routing protocols for wireless ad hoc and sensor networks, while Sec. 3 provides an introduction to opportunistic routing and discusses its theoretical properties w.r.t. data gathering. Next, in Sec. 4, we show that the difficulty of efficient opportunistic routing lies in coordinating the actual receivers of a packet to avoid duplicates. For this purpose, we propose a coordinated anypath routing scheme, called CA-Path. Our simulation results show that our scheme is indeed able to minimize retransmissions when dealing with lossy links. In Sec. 5, we focus on minimizing energy consumption and show how CA-Path may be modified for this metric. Once again, our simulation results show that good energy savings may be expected with lossy links. We finally conclude in Sec. 6 and point out future work.

2 Routing in Wireless Ad Hoc and Sensor Networks

In ad hoc wireless networks, the limited range of radio signals forces long-distance communications to be *multi-hop*, *i.e.*, intermediate nodes between the source and the destination have to relay packets [8]. Since nodes have no *a priori* knowledge of the network topology, they have to discover it. The general idea is that nodes somehow announce their presence and monitor network traffic to learn about other nodes. After some time, each node should know about at least one route to reach each other node. Unicast, broadcast, and multicast are the typical communication primitives of ad hoc networks.

A data gathering WSN is a special case of an ad hoc network, typically comprising a large number of data sources, but only a small number of data sinks, or even just a single one [9]. The responsibilities of the routing protocol are thus limited to ensuring data-flow to a very small number of destinations, as opposed to all other nodes in the network. Most often, data-gathering networks are abstracted as sink-rooted trees.

A large number of routing protocols exists for both ad hoc and sensor networks. These protocols are commonly classified according to various criteria, for instance, whether route discovery is reactive (*e.g.*, DSR [10]), or proactive (*e.g.*, OLSR [4]). Orthogonally, protocols may be singlepath, such as MintRoute [6] and the Collection Tree Protocol [11], or multipath (*e.g.*, Maximum Lifetime Routing [12]). Many more classification criteria may, in fact, be defined [13].

Nevertheless, all of these criteria fail to highlight the fact that forwarding packets is always performed in the same way: the sender selects a single next hop according to a given metric, and packets are sent exclusively to that node. This behavior actually stems from the well-established wired networks and, as we will elaborate in the next section, may not be ideally suited for *wireless* networks.

3 Opportunistic Routing for Data Gathering

As mentioned in the introduction, opportunistic routing views the routing problem with a different philosophy, compared to traditional algorithms. While choosing a next hop *a priori*, *i.e.*, before actually sending a packet, may be well suited for wired networks (in which losses are almost inexistent), this is not necessarily the case for wireless networks. Due to the broadcast nature of the wireless channel, packets are, in fact, typically transmitted to all nearby nodes[2], regardless of any chosen next-hop. Hence, finalizing the next hop routing decision only *after* sending a packet to a set of possible relays may be a better solution.

For instance, in Fig. 1, when Node E has to send a packet to the *Sink*, its three possible next hops are A, B, and C. By preselecting one of them, the probability that the chosen node will not receive the packet—and thus the retransmission probability—is $1 - 0.4 = 0.6$. Node E, however, should not have to care about which node will be the actual next hop. Much rather, it could simply broadcast the packet to the *candidate relays* A, B, and C. Subsequently, an agreement will have to be reached, regarding which node(s) will forward the packet. In this case, the probability that at least one of the prospective relays receives the packet is $1-(1-0.4)^3 = 0.784$. The retransmission probability of the packet is now equal to only 0.216, much lower than the original 0.6. Moreover, even if one of the links was very lossy, opportunistic routing could make use of it, resulting in a natural load balancing between A, B, and C. Most traditional routing algorithms would use only the better links in such a scenario.

Generalizing the Bellman-Ford algorithm, the cost of an opportunistic route comprises two components: the *anycast link cost* (ALC), which is the cost to reach the *candidate relay set* (CRS), and the *remaining path cost* (RPC), which is the cost to get from the candidate relay set to the destination. Following our previous example, the ALC is the cost to send a packet from E to the set $\{A, B, C\}$, and the RPC is the cost to send the packet from this set to the *Sink*.

[2] We assume customary omnidirectional antennas.

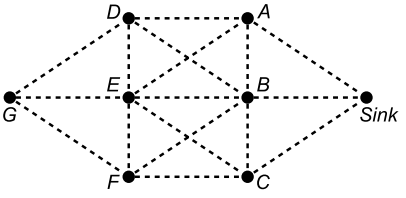


Fig. 1. Topology of a wireless network. All links have a delivery probability of 0.4.

As it is our aim to reduce the number of retransmissions, we draw upon the expected transmission count metric (ETX) [14]. In the following, the term "cost" will thus always imply the ETX metric.

3.1 Anypath Routing

Anypath routing provides a general way of computing the cost to reach a given destination [7]. To find the *shortest anypath route* (SAR) between two nodes, we assign a CRS to each node, such that the expected cost of forwarding packets via this set to the destination is minimized. Anypath routing provably computes the optimal set for each node to reach a certain destination. For this, Dubois-Ferrière *et al.* have proposed the following three equations [7].

Node i's anycast link cost, which accounts for reaching at least one node in the CRS J, is defined as

$$\mathrm{ALC}_{iJ} = \frac{1}{p_{iJ}} = \frac{1}{1 - \prod_{j \in J} (1 - p_{ij})}, \tag{1}$$

where p_{ij} is the probability of successful transmission between nodes i and j, and p_{iJ} is the probability of successful transmission between i and at least one node in the set J.

The remaining path cost of node i's CRS J is defined as

$$\mathrm{RPC}_{iJ} = \frac{C_1 p_{i1} + \sum_{j=2}^{|J|} C_j p_{ij} \prod_{k=1}^{j-1} (1 - p_{ik})}{1 - \prod_{j \in J} (1 - p_{ij})}, \tag{2}$$

assuming that the nodes in J are sorted by their cost, *i.e.*, $C_1 < C_2 < \ldots < C_{|J|}$. The numerator of this equation represents the probability of a packet being received by a particular node in the CRS J and not being received by any node with a lower cost to reach the destination, while the denominator accounts for the probability that at least one node in J has received the packet.

The cost of the shortest anypath route is defined as

$$C_i^{\mathrm{SAR}} = \min_{J \in 2^{N(i)}} \left(\mathrm{ALC}_{iJ} + \mathrm{RPC}_{iJ} \right), \tag{3}$$

where $N(i)$ is the set of neighbors of node i, and $2^{N(i)}$ is the power set of $N(i)$. Due to the aforementioned sorting of J, the search space is only of size $n = |N(i)|$, as opposed to the $2^{|N(i)|} - 1$ non-empty subsets of i's neighbors. If it is not beneficial to add neighbor j to the CRS, then it cannot be beneficial to add neighbor $j + 1$, with cost greater than that of j. Furthermore, the cost of the shortest anypath route will never be higher than that of the shortest singlepath route, since the set of opportunistic routes between two nodes includes all singlepath routes.

In examining these equations, an inherent tension between ALC and RPC becomes apparent. The ALC is minimized by taking the entire neighborhood as candidate relay set; on the contrary, the RPC is minimized by choosing only the one neighbor with the least cost to reach the destination. Note also that these equations do not consider the acknowledgment cost, although acknowledging packets is mandatory to prevent losses.

3.2 Anypath Routing for Data Gathering

Anypath routing has been specifically targeted towards ad hoc networks, in which any pair of nodes may wish to communicate. This stands in contrast to data gathering WSNs, in which communication occurs only between the sensor nodes and the sink. However, as long as there is no in-network data processing (*e.g.*, aggregation), data gathering represents a special case, in which computational overhead and memory footprint are both reduced, as each node must maintain the optimal CRS for only a single destination.

Returning to the example network of Fig. 1, we now compare the cost of the shortest singlepath route to that of the shortest anypath route for the particular case of Node E.

For both singlepath and anypath, the cost to send a packet from A, B, or C to the *Sink* is

$$C_A = C_B = C_C = \frac{1}{0.4} = 2.5\,.$$

Singlepath: The cost to send a packet from E, via A (or via either B or C), to the *Sink* is

$$C_E = \frac{1}{0.4} + C_A = 5.0\,.$$

Anypath: The cost to send a packet from E, via the CRS $J(E) = \{A, B, C\}$, to the *Sink* is

$$\begin{aligned} C_E &= \mathrm{ALC}_{E,\{A,B,C\}} + \mathrm{RPC}_{E,\{A,B,C\}} \\ &= \frac{1}{1-(1-0.4)^3} + \frac{2.5{\cdot}0.4 + 2.5{\cdot}0.4{\cdot}(1-0.4) + 2.5{\cdot}0.4{\cdot}(1-0.4)^2}{1-(1-0.4)^3} = 3.78\,. \end{aligned}$$

Table 1, providing the results for the entire network, clearly shows the potential of anypath routing. Note that the given network is an intentionally simple

Table 1. Comparison of singlepath and anypath routing w.r.t. Fig. 1

Node	Singlepath	Anypath	Improvement
A	2.50	2.50	0.0%
B	2.50	2.50	0.0%
C	2.50	2.50	0.0%
D	5.00	4.01	19.8%
E	5.00	3.78	24.5%
F	5.00	4.01	19.8%
G	7.50	5.17	31.1%
Total	30.00	24.46	18.5%

example; the advantage of anypath becomes more pronounced as the number of hops increases. Note also that we focus on shortest singlepath routing for comparison. Other protocols would show similar results, as all of them use a single next hop and would thus face the same losses.

4 Coordinated Anypath Routing

So far, we have implicitly assumed the existence of a mechanism ensuring that only one receiver—the one with the lowest cost—will indeed forward a packet. Without such a mechanism, many duplicate packets will appear, resulting in substantial growth of the energy consumption. For instance, considering Fig. 1, the full cost of sending a packet from Node E, via the CRS $\{A, B, C\}$, to the *Sink* would be $C_E = 5.1$, even more than the cost of the shortest singlepath route. This cost accounts for all possible receiver scenarios: (i) one candidate receives, (ii) two candidates receive, and (iii) all three candidates receive (and forward) the packet. Hence, coordinating the actual receivers is mandatory.

4.1 Receiver Coordination

A few solutions for receiver coordination have been proposed. While cooperative diversity schemes [15] are information-theoretically interesting, they are incompatible with today's WSN hardware. RTS/CTS-based methods (*e.g.*, MAC-layer anycasting [16]) require additional messages to be sent, that can quickly outweigh the gains obtained thanks to opportunistic routing.

Extremely Opportunistic Routing (ExOR), designed for throughput maximization [1], comprises an overhearing-based coordination scheme. To choose the effective next hop, the sender includes in its packets a prioritized list of the CRS members. Next, the receivers send their acknowledgments (ACKs) in a staggered fashion (see Fig. 2), based on each node's position in the aforementioned list. As the nodes listen to each other, they include, in their own ACK, the ID of the highest-priority actual receiver they know about—possibly, their own ID. Then, all nodes believing to be the highest-priority receiver further relay the packet. The original sender of the packet considers it successfully forwarded

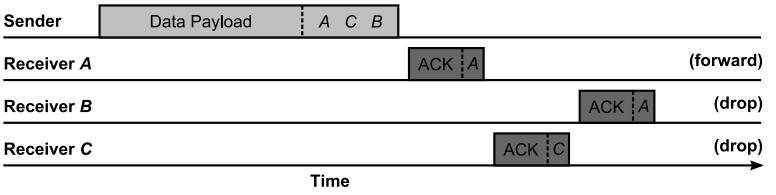


Fig. 2. The ordered list of intended receivers, sent as part of each packet, fosters collisions avoidance: ACKs are sent in a staggered fashion, rather than concurrently. Nodes include the ID of the highest-priority actual receiver they know about (by overhearing) in their own ACK. This example assumes that A and C cannot hear each other, but B can communicate with A and with C. Thus, both B and C learn about a higher-priority receiver, and drop their packets accordingly, while A will forward its packet.

as soon as it receives one ACK. Obviously, the emergence of multiple forwarders is not entirely eliminated, as it is not guaranteed that receivers are sufficiently able to overhear each other.

ExOR has been designed to increase throughput, not for energy-efficient data gathering, and we cannot use it directly for our purpose. However, as it is our goal to reduce retransmissions, we have chosen to follow the same idea for coordinating receivers. Not only does it avoid additional transmissions (ACKs are anyhow mandatory to prevent losses), but it relies solely on adding a few bytes to packets, which has little impact on transmissions (see, for instance, Fig. 3 in Sec. 5, showing the power consumption of a typical sensor mote).

4.2 Coordinated Anypath Routing for Data Gathering

Based on the cost of the shortest anypath route, given in Equation (3), the cost of the shortest *coordinated* anypath route (SCA-Path) is equal to

$$C_i^{\text{SCA-Path}} = \min_{J \in 2^{N(i)}} \left(\text{ALC}_{iJ} + \text{AC}_{iJ} + \text{RCC}_J + \text{RPC}_{iJ} \right), \tag{4}$$

where AC_{iJ} is the acknowledgment cost of CRS J at node i, and RCC_J is the cost of coordination among the nodes in CRS J.

As we are using a purely overhearing-based coordination approach, the RCC can immediately be set to zero:

$$C_i^{\text{SCA-Path}} = \min_{J \in 2^{N(i)}} \left(\text{ALC}_{iJ} + \text{AC}_{iJ} + \text{RPC}_{iJ} \right), \tag{5}$$

where the respective definitions of ALC_{iJ} and RPC_{iJ} remain as given in Sec. 3.

To define AC_{iJ}, we must introduce some additional notation. Let $\boldsymbol{A} \in \{0,1\}^{|J(i)|}$ be a random vector representing the outcome of the current transmission. Let $\boldsymbol{A}_j = 1$ iff $j \in A(i)$, where $A(i)$ is the set of actual receivers; hence, $A(i) \subseteq J(i)$. Finally, $a : \{0,1\}^{|J(i)|} \to \mathbb{R}$ is the function, which assigns—to a specific realization of $\boldsymbol{A}$—the acknowledgment cost, with $a(\mathbf{0}) = 0$. AC_{iJ} is thus the expected cost of acknowledging a packet by a certain set to actual receivers, conditional on at least one node in the CRS receiving the packet:

Table 2. Comparison of singlepath and CA-Path routing w.r.t. Fig. 1

Node	Singlepath	CA-Path	Improvement
A	5.00	5.00	0.0%
B	5.00	5.00	0.0%
C	5.00	5.00	0.0%
D	10.00	9.69	3.1%
E	10.00	9.69	3.1%
F	10.00	9.69	3.1%
G	15.00	14.38	4.2%
Total	60.00	58.44	2.6%

$$\mathrm{AC}_{iJ} = E\left[a(\boldsymbol{A}) | \sum_{k\in J} \boldsymbol{A}_k > 0\right] = \frac{E\left[a(\boldsymbol{A})\right]}{P\left(\sum_{j\in J} \boldsymbol{A}_j > 0\right)} = \frac{\sum_{A\in 2^J} P(A)\, a(A)}{1 - \prod_{j\in J}(1-p_{ij})}, \quad (6)$$

where 2^J is the power set of J, and $P(A)$ is the probability that the set of actual receivers is indeed A:

$$P(A) = \prod_{j\in A} p_{ij} \cdot \prod_{j\in J\setminus A} (1-p_{ij})\,, \quad (7)$$

and $a(A)$ is the cost of acknowledging a packet received by all nodes in A:

$$a(A) = \sum_{j\in A} \frac{1}{p_{ji}}\,. \quad (8)$$

We call this coordinated anypath routing scheme *CA-Path*. From hereon, singlepath costs will include acknowledgment costs, just as CA-Path costs do. Note that the optimal CRS determined by CA-Path is likely to be smaller than that found with anypath routing, as a larger CRS incurs a higher acknowledgment cost. Furthermore, similar to anypath routing, the cost of the shortest CA-Path route will never be higher than that of the shortest singlepath route.

Returning to Fig. 1, we now compare the cost of the shortest singlepath route to that of the shortest CA-Path route.

For both schemes, the cost to send a packet from A, B, or C to the *Sink* is

$$C_A = C_B = C_C = \frac{1}{0.4} + \frac{1}{0.4} = 5.0\,.$$

Singlepath: The cost to send a packet from E, via A (or via either B or C), to the *Sink* is

$$C_E = \frac{1}{0.4} + \frac{1}{0.4} + C_A = 10.0\,.$$

CA-Path: The cost to send a packet from E, via the CRS $J(E) = \{A, B\}$, to the *Sink* is

$$\begin{aligned} C_E &= \mathrm{ALC}_{E,\{A,B\}} + \mathrm{AC}_{E,\{A,B\}} + \mathrm{RPC}_{E,\{A,B\}} \\ &= \frac{1}{1-(1-0.4)^2} + \frac{2 \cdot \frac{1}{0.4} \cdot 0.4 \cdot (1-0.4) + \left(\frac{1}{0.4} + \frac{1}{0.4}\right) \cdot 0.4^2}{1-(1-0.4)^2} \\ &+ \frac{5.0 \cdot 0.4 + 5.0 \cdot 0.4 \cdot (1-0.4)}{1-(1-0.4)^2} = 9.69\,. \end{aligned}$$

This is the cost of the shortest CA-Path route from Node E to the *Sink*. Note that this optimal CRS is smaller than the one resulting from the original anypath equations. Table 2 provides the results for all the nodes of Fig. 1. Although we are now considering acknowledgment costs, CA-Path still has an advantage.

4.3 Theoretical Bounds

Let us assume a network, in which all links have the same delivery probability p. For such a network, we can determine the threshold probability, below which selecting multiple next hops with CA-Path provides better results than singlepath. Let us consider Node D of the network shown in Fig. 1: its two possible next hops towards the *Sink* are A and B. Since all probabilities are equal, D obviously cannot consider E and G as candidate next hops. The question, of whether it is better to use two next hops instead of one, reduces to solving the following inequality:

$$C^{\text{CA-Path}}_{D,\{A,B\}} < C^{\text{SP}}_{D,\{A\}}. \tag{9}$$

In other words, for which values of p can CA-Path decrease the cost compared to singlepath? Developing this equation, we find

$$\frac{1}{1-(1-p)^2} + \frac{2 \cdot \frac{1}{p} \cdot p \cdot (1-p) + \frac{2}{p} \cdot p^2}{1-(1-p)^2} + \frac{\frac{2}{p} \cdot p + \frac{2}{p} \cdot p \cdot (1-p)}{1-(1-p)^2} < \frac{2}{p} + \frac{2}{p}\,,$$

which solves to $p < 0.5$. Thus, when $p \geq 0.5$, CA-Path reduces to singlepath. Following similar reasoning, choosing three or even four next hops is interesting only when $p < 0.38$ or $p < 0.31$, respectively. This explains why, in Fig. 1, where $p = 0.4$, Node E uses only two candidate relays.

4.4 Simulation Results

In order to concentrate on the networking aspects, and avoid issues unrelated to CA-Path, we have developed our own open-source simulation tool[3], rather than relying on one of the existing heavyweight frameworks. The ns-2 simulator[4], for

[3] http://rr.epfl.ch/19/

[4] http://nsnam.isi.edu/nsnam/

Table 3. ETX performance of CA-Path for different types of networks: n is the number of nodes and d the average maximum number of hops to the sink

Network	Singlepath	CA-Path	Improvement
$n = 20$, $d = 2.6$	250.80	241.68	3.6%
$n = 35$, $d = 3.6$	566.67	540.76	4.6%
$n = 50$, $d = 4.4$	956.75	906.80	5.2%
$n = 100$, $d = 6.1$	2663.33	2507.12	5.9%
$n = 250$, $d = 9.4$	10152.46	9483.79	6.6%
$n = 500$, $d = 12.9$	27915.92	25982.50	6.9%

instance, entails great effort for an in-depth understanding of the interactions between its numerous components. Moreover, it requires so many modifications and add-ons for simple primitives (*e.g.*, broadcasting a packet to all neighbors) that results may not be trustworthy in the end.

Each result provided in this section is an average over 500 generated network topologies. For each topology, nodes are uniformly and randomly placed in a square area, whose size is determined by the number of nodes and the average node degree, which is fixed at 10. A single sink is present in each network. Non-connected topologies are discarded and regenerated as necessary. Link quality estimation is out of the scope of this paper; we assume that a neighborhood discovery protocol is in charge of it. A simple solution, such as counting sequence numbers, is sufficient for this purpose [2].

To limit computational complexity, the maximum CRS size in is set to three, *i.e.*, nodes select at most three next hops, even if more would further decrease their cost. If we assume that node identifiers are stored on a single byte, this leads to a maximum overhead of three bytes per data packet, which is negligible w.r.t. energy consumption. More on this subject is elaborated in the next section.

As we have pointed out previously, using CA-Path is obviously sensible only when links are lossy. In this case, the number of additional ACK transmissions is less than the number of data packet retransmissions and thus leads to a smaller ETX. On the contrary, when links are strong, CA-Path reduces to singlepath. Hence, for the results presented in this section, all links in the generated networks have a delivery probability of $p = 0.25$.

Table 3 provides the average overall ETX for various network sizes. We have considered many scenarios: small, medium, large, and very large networks. Most current WSN deployments are rather small-scale (*e.g.*, LUSTER [17], SensorScope [2]), but larger scenarios are envisioned for the near future. As expected, savings increase with the average distance to the sink, since CA-Path is able to save a few transmissions over singlepath at each hop. In fact, the last hop to the sink limits the savings, since it is cheaper for many nodes to communicate exclusively with the sink. In this case, CA-Path resumes to singlepath. Overall, results are promising for all kinds of networks, as savings range from 3.6% for small networks of 20 nodes to 6.9% for very large networks of 500 nodes. This clearly shows the potential of CA-Path when dealing with lossy links.

5 Minimizing Energy Consumption

So far, we have only considered the problem of minimizing the number of transmissions, while—in the real world—minimizing energy consumption is generally more interesting. These two metrics are of course correlated, but even with a fixed transmission power, sending packets of different sizes results in different energy consumption.

To determine the respective costs of ACKs and data packets, we have measured the power consumption of a TinyNode sensor mote[5], while transmitting packets of different payload lengths (see Fig. 3). The power consumption when sending a 0-byte payload is not zero, because each payload is preceded by a network header. Moreover, the radio precedes each packet with a specific pattern of bits, so that receivers can detect the beginning of incoming packets[6]. Based on our measurements, the duration of transmission—and thus the power consumption—may be approximated as

$$C(l) = 0.1043 \cdot l + 1.966, \tag{10}$$

where l is the payload length, in bytes. According to this equation, sending a 28-byte data packet costs $C(28) = 4.89$ while a 2-byte acknowledgment costs $C(2) = 2.17$. The ratio between the two costs is 0.44, which we approximate by 0.5, making ACKs a bit more costly than in reality. Thus, in the following, we assume that an acknowledgment consumes half as much energy as a data packet.

Considering these relative costs has no impact on singlepath, which will always select the same path to send packets to the sink. With CA-Path, things are different, as the CRS size impacts the acknowledgment cost. To take this into account, we must modify Equation (8), such that $a(A)$ is now equal to

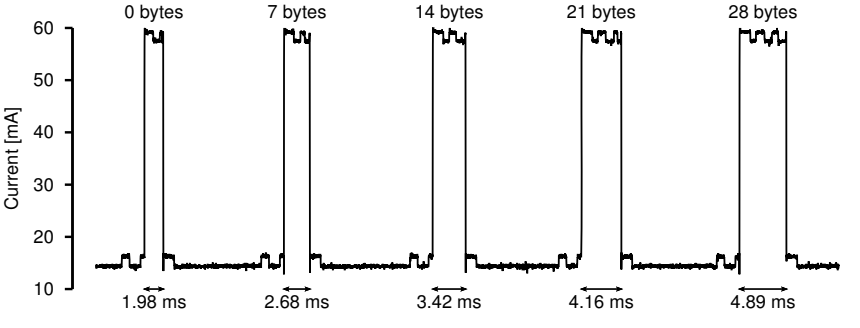


Fig. 3. Power consumption of a TinyNode mote when sending packets of various payload lengths. The transmission power is set to 15 dBm.

[5] http://tinynode.com/
[6] Note that this is not related to a low power listening mechanism.

$$a(A) = \sum_{j \in A} \frac{0.5}{p_{ji}}, \tag{11}$$

while the other cost functions remain unchanged.

5.1 Implementation Issues

CA-Path may be implemented as a purely overhearing-based, proactive routing protocol, *i.e.*, no additional messages are required for route maintenance. Nodes must keep an up-to-date table of their neighbors' costs and include their own cost to reach the sink, together with the chosen list of intended receivers, in each data packet. Any node, overhearing this information, can update its neighborhood table accordingly. To get the process started, the sink must send out beacons, advertising its own cost of zero.

In order to find the best CRS, each node needs to examine all possible subsets of its neighbors. Let us assume that d is the number of neighbors and m is the maximum CRS size ($1 \leq m \leq d$) we wish to consider. With the full-fledged scheme presented above, the number of candidate relay sets to examine is

$$\sum_{k=1}^{m} \binom{d}{k},$$

which results in a time complexity of $O(d^m)$. As savings with CA-Path may occur only when $m \geq 2$, complexity quickly becomes a problem on today's motes.

To overcome this issue, we propose the following greedy heuristic: each node evaluates its own cost w.r.t. all singleton CRSs and sorts those accordingly. Now, the node sets its CRS to be the least expensive singleton and tries to merge it with the next best one. If that decreases its cost, the node sets its CRS to be these two nodes and then tries to merge it with the third best singleton. The process is repeated until (i) the cost of the current node no longer decreases or (ii) the CRS has reached its maximum size.

This heuristic will consider at most $d+m-1$ CRSs, leading to a complexity of $O(d)$, which is much better suited to the capabilities of current sensor motes, as well as being more scalable. As we will experimentally show below, the heuristic is a good approximation of the exhaustive search for the optimal CRS.

5.2 Simulation Results

To evaluate our protocol in terms of energy consumption, we have used the same simulation tool as before, incorporating the modified equation for CA-Path. All parameters remain the same, except for the generated topologies: instead of using a uniform delivery probability, each link is assigned a random probability p, such that $0.1 \leq p \leq 0.4$. This modification is needed to evaluate the heuristic (denoted CA-Path (H)) we have proposed above; with uniform probabilities, there is no difference to the full-fledged scheme.

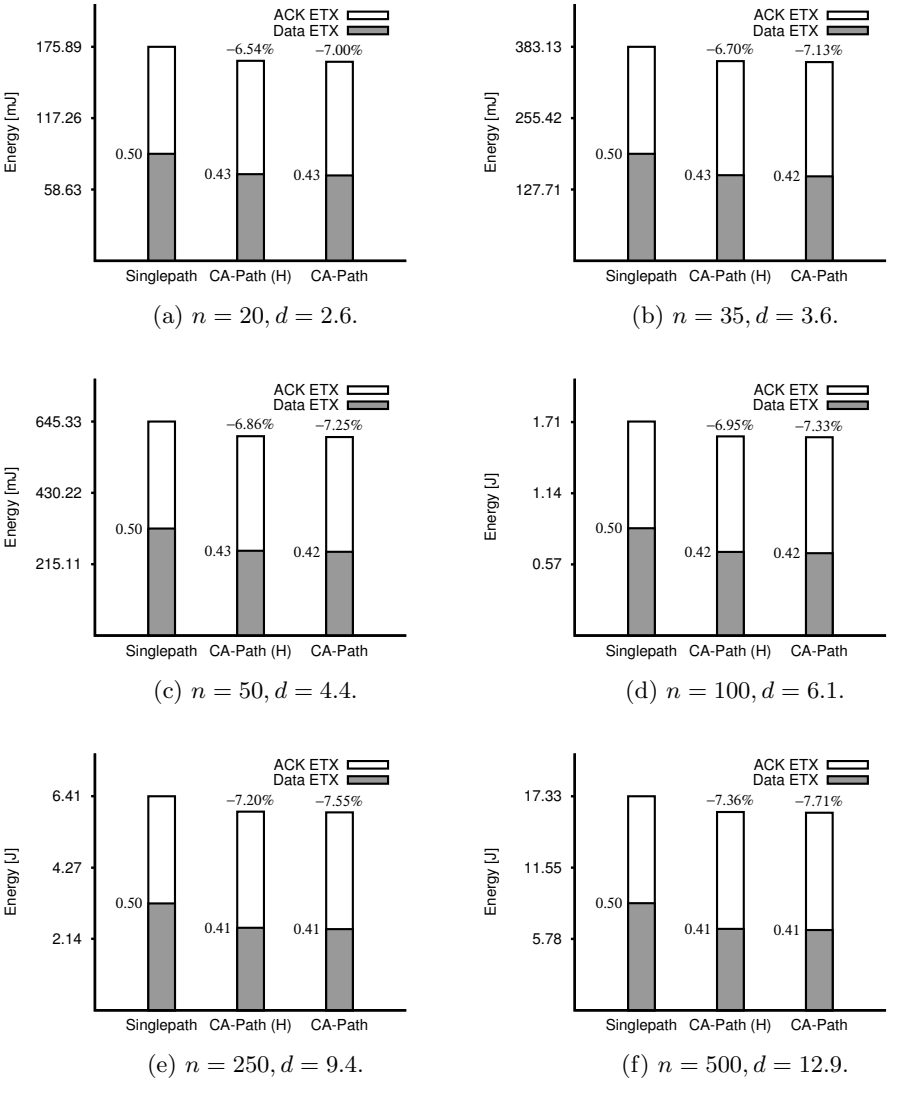


Fig. 4. Performance of CA-Path when minimizing energy consumption. The "(H)" indicates use of the heuristic instead of exhaustive search for the optimal CRS. Each bar also shows the ratio between data packet and ACK transmissions.

Figure 4 depicts the average overall expected energy consumption for the various network sizes we have considered. This consumption includes the transmission (and acknowledgment) of one data packet from each node to the sink. For instance, when $n = 20$, using singlepath leads to a global consumption of 175.89 mJ at each transmission cycle. To compute these values, we made use of the TinyNode power consumption data illustrated in Fig. 3. Sending a

28-byte data packet draws 60 mA at 3.3 V for 4.89 ms, leading to a consumption of 0.97 mJ. Similarly, sending an ACK consumes 0.43 mJ.

From these results, we can see that the ratio of energy saved by CA-Path ranges from 7.00% for small networks of 20 nodes to 7.71% for large networks of 500 nodes. Overall, the greedy heuristic provides results close to the optimal ones, always within half of a percentage point. This is encouraging, since the complexity of the heuristic is linear, making it very scalable.

Figure 4 also provides the distribution of the expected number of transmissions between data packets and acknowledgments. For singlepath, the ratio is always 0.5, since each data packet triggers one ACK. For CA-Path, we can observe that the transmission load is shifted from data packets to ACKs, and the ratio goes down to 0.41 for the 500-node networks. As a result, even when the same overall number of packets is sent, CA-Path leads to energy savings due to the smaller cost of ACKs.

6 Conclusion and Perspectives

In this paper, we have studied the potentials of opportunistic routing in data gathering wireless sensor networks. CA-Path, our implementation of a coordinated opportunistic routing scheme, is specifically designed to limit energy wasting when dealing with radio links of poor quality. We have shown that with strong links, CA-Path reduces to singlepath, while, when working in difficult environments with lossy links (*e.g.*, very long distance between nodes), CA-Path is a viable solution, leading to energy savings. With its linear time complexity, the heuristic we have proposed is especially suited to embedded sensor motes and provides results very close to exhaustive search.

The results we have shown must, however, be moderated a bit: in our simulations, routes are loop-free, and all actual receivers of a data packet can hear each other. While loops may affect any routing protocol and measures may be taken against them, non-overheard ACKs are more difficult to cope with. Due to our selection scheme, next hops should be close to each other, but of course this does not ensure that they can actually hear each other. The resulting duplicates have the potential to outweigh the savings obtained with CA-Path. We are thus planning to benchmark CA-Path by implementing it on a real sensor network. We will also study the extend of load balancing induced by CA-Path, in comparison to traditional protocols. In this area, too, good results are expected.

Finally, our results pave the way to even greater energy savings, when combined with other coordination schemes. Decreasing the coordination cost will lead CA-Path to using larger candidate relays sets and thus to lower energy consumption. One possibility could be to acknowledge multiple data packets at once, by working with a window of sequence numbers, similar to TCP. In this case, the cost of ACKs would be lowered, although at the expense of higher latency. Hence, application-specific solutions should provide the best results, overall.

Acknowledgments

This work was partially financed by the Swiss National Center of Competence in Research for Mobile Information and Communication Systems (NCCR MICS) and the European Commission under the FP6 project WASP.

References

1. Biswas, S., Morris, R.: ExOR: Opportunistic multi-hop routing for wireless networks. In: Proceedings of the ACM SIGCOMM Conference (October 2005)
2. Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M., Couach, O., Parlange, M.: Sensorscope: Out-of-the-box environmental monitoring. In: Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN) (April 2008)
3. Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M.: The hitchhiker's guide to successful wireless sensor network deployments. In: Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys) (November 2008)
4. Jacquet, P., Mühlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., Viennot, L.: Optimized link state routing protocol for ad hoc networks. In: Proceedings of the IEEE International Multi-topic Conference (INMIC) (December 2001)
5. Haas, Z.J.: A new routing protocol for the reconfigurable wireless networks. In: Proceedings of the IEEE International Conference on Universal Personal Communications (October 1997)
6. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys) (November 2003)
7. Dubois-Ferrière, H., Grossglauser, M., Vetterli, M.: Least-cost opportunistic routing. In: Proceedings of the Allerton Conference on Communication, Control, and Computing (September 2007)
8. Basagni, S., Giordano, S., Stojmenović, I.: Mobile Ad Hoc Networking. IEEE Computer Society Press, Los Alamitos (2004)
9. Estrin, D., Girod, L., Pottie, G., Srivastava, M.: Instrumenting the world with wireless sensor networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (May 2001)
10. Johnson, D.B.: Routing in ad hoc networks of mobile hosts. In: Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA) (December 1994)
11. Fonseca, R., Gnawali, O., Jamieson, K., Kim, S., Levis, P., Woo, A.: The collection tree protocol (CTP) (2006), http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html
12. Chang, J.H., Tassiulas, L.: Maximum lifetime routing in wireless sensor networks. IEEE/ACM Transactions on Networking 12(4), 609–619 (2004)
13. Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: A survey. IEEE Wireless Communications 11(6), 6–28 (2004)
14. Couto, D.D., Aguayo, D., Bicket, J., Morris, R.: A high-throughput path metric for multi-hop wireless routing. In: Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom) (September 2003)

15. Laneman, J., Tse, D., Wornell, G.: Cooperative diversity in wireless networks: Efficient protocols and outage behavior. IEEE Transactions on Information Theory 50(12), 3062–3080 (2004)
16. Choudhury, R., Vaidya, N.: Mac-layer anycasting in ad hoc networks. In: Proceedings of the ACM SIGCOMM Conference (August 2004)
17. Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., Porter, J.: LUSTER: Wireless sensor network for environmental research. In: Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys) (November 2007)

A Better Choice for Sensor Sleeping

Ou Yang and Wendi Heinzelman

Dept. of ECE, University of Rochester, Rochester, NY, 14620, USA
oyang@ece.rochester.edu, wheinzel@ece.rochester.edu

Abstract. Sensor sleeping is a widely-used and cost-effective technique to save energy in wireless sensor networks. Protocols at different stack levels can, either individually or simultaneously, make the sensor sleep so as to extend the application lifetime. To determine the best choice for sensor sleeping under different network conditions and application requirements, we investigate single layer and multi-layer sleeping schemes at the routing and MAC layers. Our results show that routing layer sleeping performs better when there is high network redundancy or high contention, while MAC layer sleeping performs better when there is low contention or in small networks. Moreover, multi-layer sleeping requires cross-layer coordination to outperform single layer sleeping under low contention. Therefore, our conclusions can not only guide the implementation of practical sensor networks, but they also provide hints to the design of cross-layer power management to dynamically choose the best sleeping scheme under different network and application scenarios.

Keywords: Sensor sleeping, Directed Diffusion, SMAC, Cross-layer coordination.

1 Introduction

Wireless sensor networks are gaining increasing attention for practical uses ranging from military surveillance to civil monitoring due to their low cost, ease of deployment and good coverage of the monitored area. However, one of the main issues preventing the ubiquitous use of wireless sensor networks is how to support an application for extended periods of time, as sensors are usually battery-powered and hence highly constrained in energy supply. Consequently, to save energy, the idea of "making sensors sleep when they are not used" has motivated much research in wireless sensor networks [1] [2] [3] [4] [5] [6] [7]. In particular, the selection of source node(s) that should transmit data to the sink(s), which is done at the application layer, allows redundant source nodes to sleep in order to save energy for later use. Topology control, which is usually performed at the network layer, creates a routing backbone for data delivery so that the remaining non-router nodes can sleep. Turning off routing nodes that are not directly involved in the delivery of data can also be done by the routing protocol. Finally, duty cycling is often employed by the MAC protocol, allowing sensors to sleep periodically to reduce their idle listening, which is energy intensive.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 134–149, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Since protocols at different stack layers can make sensors sleep, the question becomes at which layer should sleeping be used in order to extend application lifetime for a specific network and application scenario? In other words, is there a particular layer in which sleep scheduling provides the most benefit? Can sleep scheduling be done at multiple layers independently for even more gain than simply sleeping at a single layer? Or is some sort of cross-layer coordination of sleep scheduling needed to obtain the maximum benefit in terms of extending application lifetime? To examine these questions, in this paper we compare sleeping techniques employed by routing protocols and MAC protocols under various network scenarios and application requirements. Application layer sleeping schemes are currently not considered to avoid any application-specific conclusions. Thus, the contributions of this paper are:

(1) We propose a sleeping scheme for Directed Diffusion [9], which significantly improves the application lifetime by allowing nodes not involved in the transmission of data to sleep periodically. (2) We provide detailed performance comparisons of MAC layer sleeping and routing layer sleeping, under various node densities, different network scales, diverse numbers of source nodes and varying application data rates. These results show that MAC layer sleeping works better under conditions of low contention, while routing layer sleeping works better in networks with high redundancy. (3) We also examine the performance of making sensors sleep at the routing and MAC layers simultaneously, with and without cross-layer coordination. Our preliminary results show that it is necessary to employ cross-layer coordination between the layers to obtain further improvement in throughput when the contention in the network is low. (4) Our conclusions can be used to design a policy for a centralized power manager that dynamically decides which layer(s) should have control over sleep scheduling as network parameters vary or application requirements change over time.

2 Motivation and Background

Wireless sensors have stringent energy constraints since they are usually battery-powered, and oftentimes it is impractical to recharge the sensors manually due to their large-scale deployment. Hence, it is critical to employ energy-saving techniques so as to support an application for extended periods of time. Among the existing energy-saving strategies, e.g., flooding avoidance [8] [9], traffic balancing using cost functions [10] [11], data fusion [12], etc., putting sensors into the sleep state is the most widely-used and cost-effective way to prolong the application lifetime. By turning off the radio of a sensor at appropriate times, various sleeping schemes [1] [2] [3] [4] [5] [6] [7] aim to cut down on "idle listening", which in wireless sensor networks provides little benefit yet consumes almost as much power as transmitting or receiving data [13].

To save the most energy, it is intuitive to make sensors sleep whenever possible. This implies that (1) redundant source nodes may sleep when their sensed data is not required by the sink, (2) redundant routing nodes may sleep when they have no data to relay, and (3) source nodes and routing nodes that are involved

in the data delivery may also sleep when it is not their turn to transmit or receive data. Correspondingly, existing sleeping schemes are implemented in such a way that different layers perform these different tasks mentioned above.

Specifically, the application layer selects and activates only some of the source nodes to reduce energy drain while maintaining desired QoS [1]. For example, in the application of target tracking [1], all sensors within a certain distance of the target could be source nodes, but only a few are finally activated by the application layer to achieve low distortion and reduce the energy consumption.

Routing nodes, on the other hand, are usually selected and activated by the network layer in two ways. The first way is called topology control [2] [3], which aims to establish a routing backbone to guarantee the network connectivity, and consequently non-backbone nodes can sleep. Topology control also updates the backbone periodically according to the remaining energy of each sensor to balance the energy consumption. For instance, GAF [2] is a topology control protocol, which divides a network into virtual grids with only one sensor in each grid activated at any time, constituting a backbone network to route all the data. Topology control is often used in large scale, dense networks where many nodes provide redundant routes. Hence, making all of them except the activated sensors sleep can significantly extend the application lifetime.

The second way of making sensors sleep at the network layer is through routing protocols. Routing protocols [4] [5] update routes periodically, and save energy by turning off the routing nodes that are not involved in the data delivery. For example, the technique described in [4] puts sensors to sleep by monitoring routing control messages and data transmissions so that only useful routers are kept active. The technique described in [5] also utilizes the routing decisions so that sensor nodes do not wake up when they are not part of a routing path. Compared to topology control, those routing protocols with sleeping do not waste the energy of non-used routing nodes, and they are also suitable for sparse or not large scale networks where topology control performs poorly.

Once source nodes and routing paths are determined, the MAC layer can put sensors into sleep-awake cycles [6] [7] so that idle-listening can be further reduced due to the fact that the traffic load in wireless sensor networks is usually not very high [6]. For example, SMAC [6] makes each sensor broadcast its sleep-awake schedule once in a while, so that its neighboring sensors can hear it and synchronize their schedules with it. Once sensors are synchronized, they use RTS/CTS/DATA/ACK sequences to communicate during the awake period of each cycle. BMAC [7], using periodic low power listening, forces a sensor to sleep if nothing is detected on the media in the sensor's awake duration. However, the benefit of longer application lifetime provided by MAC layer sleeping accompanies lower throughput and higher latency.

As sleeping can be implemented at the application layer, network layer and MAC layer individually, questions arise as to which layer provides the most benefit for determining when a sensor should sleep, under what conditions should each sleeping scheme be chosen, and whether cross-layer coordination is needed to obtain further improvement when sleeping schemes are employed at different

layers simultaneously. The answer to these questions may not only guide us in the implementation of practical sensor networks, but they may also provide hints into the design of cross-layer power management.

3 Sleeping at Different Layers Individually and Simultaneously

As wireless sensing applications operate in diverse networking environments and have a range of QoS requirements, source node selection at the application layer is always application-specific, and hence it is hard to generalize its performance. Therefore, to avoid any application-specific conclusions, in this paper we choose a general query-based application, and we focus on more general sleeping strategies at the network and MAC layers. Specifically, we look into sleeping schemes implemented in the routing and MAC protocols, defined as "routing layer sleeping" and "MAC layer sleeping", respectively, throughout this paper. Topology control (done by the network layer) is currently not investigated. We make conclusions by comparing the performance of (1) a non-sleeping routing protocol with a non-sleeping MAC protocol, (2) a non-sleeping routing protocol with a sleeping MAC protocol, (3) a sleeping routing protocol with a non-sleeping MAC protocol, and (4) a sleeping routing protocol with a sleeping MAC protocol.

For the non-sleeping routing protocol, we choose Directed Diffusion [9], as it is specially designed for data-centric sensor networks and widely accepted as a typical routing protocol for wireless sensor networks. However, Directed Diffusion does not include any sleeping strategy. We, therefore, propose an improved Directed Diffusion with sleeping as our sleeping routing protocol (see details in Section 3.1). The sleeping Directed Diffusion performs exactly the same as the original Directed Diffusion in terms of source node discovery and routing path establishment and maintenance, but sleeping Directed Diffusion allows routing nodes to sleep during the interval between two successive routing updates, if they are not reinforced by the sink to relay data. Therefore, the application lifetime may be prolonged by getting data from another routing path once the previous path dies due to energy depletion.

For the MAC layer, we use IEEE 802.11 and SMAC as the non-sleeping and sleeping protocols, respectively. SMAC is a typical MAC protocol with duty cycled sleeping for wireless sensor networks. Except for the duty cycle, SMAC is similar to IEEE 802.11. In other words, SMAC with 100% duty cycle has the same performance as IEEE 802.11 under light traffic load. Hence, comparing the performance of the network using IEEE 802.11 and SMAC provides clear insight into the benefits and drawbacks of duty cycling the sensors at the MAC layer.

3.1 Sleeping at the Routing Layer

To understand our proposed sleeping Directed Diffusion, we first review the mechanisms of Directed Diffusion, and then we explain our implementation of sleep-awake cycles used with Directed Diffusion.

Directed Diffusion. Directed Diffusion [9] includes two phases, an exploratory phase and a reinforcement phase, which together allow a sink node to obtain desirable data from source nodes. The exploratory phase is used to discover data sources at a low data rate, while the reinforcement phase is used to pull down the desirable data at a high data rate.

In the exploratory phase, a sink starts broadcasting INTEREST packets periodically in the network. An INTEREST packet includes a description of desired data attributes and indicates the exploratory rate of this specific data (usually as low as one packet per hundreds of seconds). A node that receives an INTEREST packet floods it to all of its neighbors. Meanwhile, every node maintains a gradient table, caching each distinguishable INTEREST packet in terms of where it came from (previous hop) and what the desired data rate is as the local routing information. Each caching entry is called a gradient, which expires and hence will be removed from the gradient table after a certain time, so as to take care of topology changes or node failures. New gradients will be established by the periodic INTEREST packets flooded in the network. As a result, after some time, all the nodes, including the data sources in the network know the INTEREST, and all possible routing paths are established in a distributed manner by checking gradient information at each node. When a source node receives an INTEREST packet, it broadcasts DATA packets to all its neighbors at the exploratory rate. All the intermediate nodes cache and flood all distinguishable DATA packets and discard duplicate ones (e.g., if the time stamp is very close - less than the interval of two successive high rate DATA packets - to any received DATA packet). Finally, the sink node receives the DATA packets that it is interested in from multiple paths. Then the sink starts the reinforcement phase.

In the reinforcement phase, the sink node has a specific policy to reinforce some of the routing paths to pull down the data from the source nodes at high data rates. The policy in our experiments is to reinforce the neighbor that delivered the exploratory data first. The sink node unicasts a positive reinforcement packet, which is actually an INTEREST packet with high desired data rate (usually tens of times higher than the exploratory rate), to the selected neighbor, and then the selected neighbor forwards this positive reinforcement packet to the next hop, which is chosen by the same policy. When a source node receives a positive reinforcement packet, it starts sending back DATA packets at the requested high data rate, along the path where the positive reinforcement packet came from. Therefore, the sink node finally obtains data at the desired high rate from the reinforced path. In the case of topology changes, node failures or link quality changes, a new path may be positively reinforced. Hence, every sensor periodically checks if a negative reinforcement is needed to slow down the data delivery on the previously reinforced paths.

Directed Diffusion does not allow sensors to sleep. However, not every routing node is involved in the data delivery all the time, as only the least delayed routing path is positively reinforced. Hence, allowing redundant routing nodes to sleep may save other routing paths and contribute to a longer application lifetime when the previous ones run out of energy.

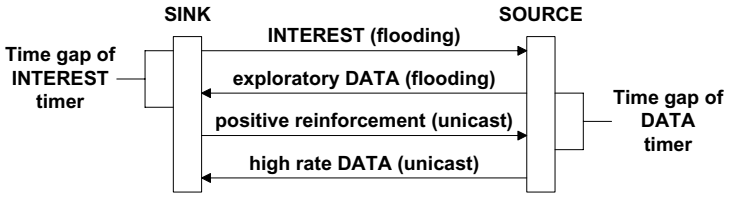


Fig. 1. Time sequences of path establishment in sleeping Directed Diffusion

Sleeping Directed Diffusion. To make routing nodes sleep when they are not involved in the data delivery, it is necessary to note the importance of INTEREST flooding and exploratory DATA flooding, and figure out the time sequences of path establishment and maintenance in Directed Diffusion.

As mentioned above, in Directed Diffusion there are two critical floodings throughout the network. One is the periodic flooding of an INTEREST packet, initiated by the sink node. All routing nodes need to be awake to forward this packet so that source nodes can finally receive it, and a gradient table can be established to route the DATA packets that are going to follow. The other flooding is exploratory DATA flooding, periodically initiated by a source node. All routing nodes need to be awake to forward those packets so that the sink node can finally receive them and start positive reinforcement based on their arrival times. As a result, for each distinguishable INTEREST (query for different data attributes), we define two timers, INTEREST timer and DATA timer, at each node, for INTEREST flooding and exploratory DATA flooding, respectively. The two timers are scheduled and fire periodically. Specifically, INTEREST timer is scheduled after an INTEREST flooding ends, and fires before the next INTEREST flooding starts, while DATA timer is scheduled after an exploratory DATA flooding ends, and fires before the next exploratory DATA flooding starts. A sensor may sleep when the two timers are pending, but it MUST wake up once either timer fires, and remain awake until the timer is rescheduled.

There is a time gap between when a timer fires and when it is rescheduled. The time gap is used to wait for the response of the corresponding flooding so as to establish a reinforced path from the sink node to the source node. As shown in Fig. 1, path establishment starts with INTEREST flooding, followed by the exploratory DATA flooding, which is then followed by positive reinforcement unicasting, and consequently followed by high rate DATA unicasting. Hence, the time gap of the INTEREST timer is used to wait for the exploratory DATA flooding if new source nodes are discovered, while the time gap of the DATA timer is used to wait for the positive reinforcement packets so that one of the source nodes can be reinforced and start delivering high rate data.

If a node does not receive any exploratory DATA packet during the time gap of its INTEREST timer as a result of no source node available or packet loss, the exploratory DATA timer is not initiated. Hence, the node follows the INTEREST timer to sleep (when it is pending) and wake up (when it expires). Otherwise, the exploratory DATA timer is initiated, and it is scheduled periodically according

to the exploratory rate. If the node receives a positive reinforcement packet during the time gap of its DATA timer, the node is located on the reinforced path (involved in the data delivery), and thus cannot sleep until it is negatively reinforced or it runs out of energy. If the node does not receive any positive reinforcement packet during the time gap of its DATA timer, the node goes to sleep when both of the timers are pending (no flooding is going on), and wakes up whenever at least one of the timers expires (at least one flooding is going on).

3.2 Sleeping at the MAC Layer

SMAC is a MAC protocol explicitly proposed for wireless sensor networks, with reducing energy consumption as its primary goal [6]. It introduces periodic sleep-awake cycles for each node to eliminate idle-listening. Used for transmitting and receiving data, the awake period of a cycle has a fixed length, which is determined by the physical layer and MAC layer parameters. The sleeping period of a cycle, instead, can be set longer or shorter, which influences the power consumption of SMAC, as well as the latency incurred by sleeping. Hence, variations in duty cycle, which is defined as the ratio of the awake period to a complete sleep and awake cycle, leads to corresponding variations in the performance of SMAC.

For the convenience of communication, and to reduce the control overhead, SMAC tries to synchronize every node, so that nodes sleep and wake up simultaneously. To achieve this, every node periodically broadcasts its schedule in a SYNC packet, so that neighbors who hear the SYNC packet start following the same schedule. However, some nodes may not hear the SYNC packets from their neighbors because they have already been running different schedules. Hence, every node must keep awake for a whole SYNC packet interval once in a while, so that different schedules of its neighbors can be heard. A node that has a different schedule from its neighbors may follow both schedules at the same time.

During the awake time, SMAC is similar to IEEE 802.11, which (1) uses RTS/CTS to solve the hidden terminal problem, (2) uses physical carrier sensing and virtual carrier sensing to avoid collision, and (3) uses RTS/CTS/DATA/ACK sequences to guarantee successful unicast transmissions. If a node fails to access the media, it goes back to sleep until the next awake period. If, on the other hand, a node successfully accesses the media, it does not sleep until it finishes the current transmission.

3.3 Sleeping at Both Routing and MAC Layers

To employ sleeping Directed Diffusion and SMAC simultaneously, we can either simply implement them as they are, at the routing layer and the MAC layer, respectively, or do cross-layer coordination between the routing layer and the MAC layer to improve the overall performance.

There are various ways to coordinate the routing and MAC layer sleeping. We implement two methods in our simulations. The first cross-layer coordination is based on priority. As sleeping Directed Diffusion puts all the nodes that are not involved in the data delivery to sleep during successive floodings, there is no need to make those nodes wake up at the MAC layer according to its duty cycle,

as no data needs to be transmitted. Hence, sleeping Directed Diffusion should have higher priority than SMAC to schedule the nodes. In other words, SMAC only effectively schedules a node when sleeping Directed Diffusion needs to keep this node active. The second cross-layer coordination is differentiation between routing layer sleeping and energy depletion. As SMAC updates a neighbor list at each node by recognizing SYNC packets sent by its neighbors for a given period of time, long sleeping time of a node scheduled by sleeping Directed Diffusion may make the node's neighbors mistakenly drop its information from their neighbor lists, as no SYNC packet is sent from this node during its sleeping time. We, therefore, make a SYNC packet bear the remaining energy of its sender, so that the receiving nodes can easily tell the status of the sender, and remove the sender from its neighbor list only when it is running out of energy.

4 Simulations and Discussions

In this section, we analyze the pros and cons of making sensors sleep at individual layers under different network scenarios and application requirements. We also give some preliminary results on the necessity of cross-layer coordination when sensors sleep at both layers. Two metrics are considered. The first metric is throughput, which is the total number of packets received by the sink node. In general, the higher the throughput, the more information is collected at the sink, which corresponds to a longer application lifetime. However, a high throughput does not necessarily imply a good data delivery ratio. Therefore, we define the second metric, data delivery ratio, as the throughput divided by the number of packets the sink node should ideally receive. Ideally, the sink node receives as many packets as generated by a single reinforced source node during the whole data delivery period. The higher the data delivery ratio is, the fewer packets are lost. In general, for a given application and network deployment, we desire both throughput and data delivery ratio to be high.

We use ns-2.32 [14] to simulate the performance of all combinations of a non-sleeping/sleeping routing protocol (Directed Diffusion/sleeping Directed Diffusion) and a non-sleeping/sleeping MAC protocol(IEEE 802.11/SMAC[1]). We assume that (1) sensors are static and randomly distributed in a given area, (2) there is only one sink node and more than one source node, (3) the sink node has infinite power supply, while other nodes have 22J initial energy, (4) each node's power consumptions in transmitting, receiving, and idle status are set the same at 50mW, based on measurements of CC1000, a radio chip for MICA2 motes, and CC2420, a radio chip for IEEE 802.15.4 [13], (5) the size of an application layer data packet is 64 bytes, (6) MAC layer bandwidth is 2Mbps, and (7) the communication range of a sensor is 250m.

For Directed Diffusion, we assume that the sink node floods an INTEREST packet every 30s. Each gradient entry in the gradient table is valid for 50s. The exploratory rate is 1 packet per 100s. Negative reinforcement check is executed every 6 high rate DATA packet intervals. For sleeping Directed Diffusion, either

[1] Adaptive listening [9] is used in the simulation of SMAC.

timer has a 6s time gap. For SMAC, the size of a DATA packet is 50 bytes, the size of a SYNC packet is 9 bytes, and the size of other control packets, like RTS, CTS, and ACK, are 10 bytes. A SYNC packet is sent by each node every 10 duty cycles.

We first look into the performance of individual layer sleeping schemes, namely sleeping Directed Diffusion and SMAC, and then compare their performances under different situations. Finally, we show the performance differences of sleeping at both routing and MAC layers, with and without cross-layer coordination. By default, 30 nodes are randomly distributed in an 800mX800m area. There is 1 sink node and 5 source nodes. A reinforced source node generates 3 application layer packets per 10s. For each scenario, 10 topologies are generated, and the results are averaged over 20 simulations, except as noted.

4.1 Performance of Single Layer Sleeping

Fig. 2 shows the throughput of single layer sleeping schemes over time in one simulation. IEEE 802.11 is used as the common MAC protocol in the simulation of sleeping Directed Diffusion, while Directed Diffusion is used as the common routing protocol in the simulation of SMAC. As we can see, both sleeping Directed Diffusion and SMAC can significantly improve the data delivery period, and hence the overall throughput. However, sleeping Directed Diffusion has a "QoS pause" between 740s to 800s, as the throughput remains the same during this period of time. Since a new path reinforcement is always triggered by an exploratory DATA flooding, there might be a gap during which the old path has died but a new path has not been established, and therefore no DATA packets are delivered to the sink. Directed Diffusion does not have a QoS pause because all the nodes in the network die at the same time (a sensor's power consumption in transmitting, receiving and idling are the same), thus no redundant routing paths or source nodes are available to use. As a result, Directed Diffusion leads to a short data delivery period. Moreover, compared to ideal receiving, sleeping Directed Diffusion has almost the same increasing slope in throughput except QoS pauses, but SMAC always has lower throughput, because SMAC suffers from more contention and hence more collisions than IEEE 802.11.

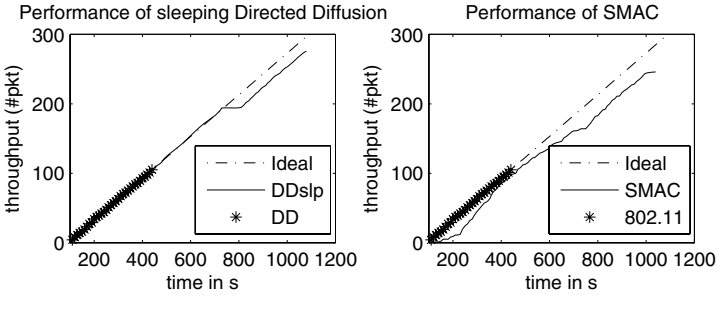


Fig. 2. Performance of sleeping Directed Diffusion and SMAC

4.2 Performance Comparisons of Individual Layer Sleeping

In this section, we examine the performance of sleeping at individual layers. For simplicity, we mention the combination of Directed Diffusion and IEEE 802.11 as DD802, the combination of Directed Diffusion and SMAC as DDSMAC followed by a specific duty cycle, and the combination of sleeping Directed Diffusion and IEEE 802.11 as DDslp802.

Varying Node Density. In this experiment, we vary the node density. Fig. 3 shows the performance rendered by 7 schemes under the deployment of 10 nodes, 20 nodes, 30 nodes, 40 nodes and 50 nodes, within the fixed area. As we can see, without sleeping, DD802 performs the same in both throughput and data delivery ratio, no matter what the node density is. This provides a baseline to judge the performance of all the sleeping schemes. Specifically, DD802 has a throughput of 105 packets on average with 100% data delivery ratio. However, its data delivery period is short in the absence of any sleeping technique.

DDSMAC, on the other hand, shows a diversity of performance under different duty cycles and different node densities. For a given node density, generally, the lower the duty cycle is, the longer alive time every sensor in the network can have, which is beneficial to receiving more packets at the sink node. However, a low duty cycle also leads to severe contention and consequently higher probability of collision. One one hand, collisions may affect the SYNC packet exchange, so that some sensors cannot communicate with each other, and hence have to drop packets. On the other hand, consistent collisions may either lead to packet drop once the packet is retransmitted up to the limit, or incur long packet delay, so that negative reinforcement may be initiated by Directed Diffusion, and then the data delivery path is cut off. Therefore, lower duty cycles always have lower data delivery ratio, but lower duty cycles may not necessarily have high throughput. As shown in Fig. 3, the best duty cycle for DDSMAC in terms of throughput varies according to the node density, which reflects the contention in the network. As the node density increases, the duty cycle with highest throughput increases from 10% for 10 nodes to 40% for 50 nodes. Note that when the node density is very high, more than 40 nodes in our experiment, DDSMAC under all duty cycles performs worse than DD802 in both throughput and data delivery ratio. Therefore, it is not worth sleeping only at the MAC layer in this case. For a given duty cycle, both throughput and data delivery ratio decrease as the node density increases. This can be explained by the fact that high node density causes high contention.

On the contrary, DDslp802 has higher throughput as the node density increases. Higher node density implies more routing redundancy. DDSMAC wastes the routing redundancy, but DDslp802 takes advantage of this added redundancy. Specifically, DDSMAC wakes up redundant routing nodes the same as it wakes up reinforced routing nodes. Hence when a reinforced routing path runs out of energy, all the other redundant paths run out of energy as well. The maximum throughput of DDSMAC is upper-bounded by the the total number of packets that can be generated by a single source node. DDslp802, however,

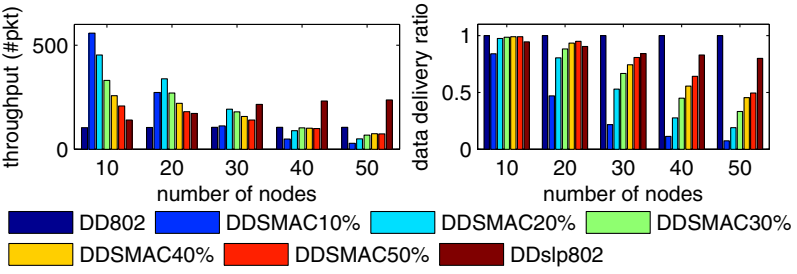


Fig. 3. Performance comparison under varying node density

saves the energy of those redundant paths by allowing redundant routing nodes to sleep, so that when one path dies, another path can be used to deliver packets. The higher the node density is, the more redundant paths will be available to improve the data delivery period as well as the throughput. However, DDslp802 suffers from QoS pauses. The more often it changes to a new routing path, the more chances it introduces a period of QoS pause. Hence, DDslp802 has worse data delivery ratio as the node density increases. However, the data delivery ratio of DDslp802 decreases much slower than the data delivery ratio of DDSMAC. Fig. 3 also shows that QoS pause impairs the data delivery ratio more than SMAC unreliability at low node density, but it impairs the data delivery ratio less than SMAC unreliability at high node density.

Due to space limitations, the standard deviations of throughput and data delivery ratio are not shown in the figures. However, we observed that DD802 and DDslp802 have very small standard deviations for both throughput and data delivery ratio, while DDSMAC has larger standard deviations as the node density increases or as the duty cycle decreases. Similar results are observed in all the experiments throughout the paper.

Varying Network Scale. In this experiment, we fix the node density, but vary the network scale. 17, 23, 30, 38 and 47 nodes are placed in a 600mX600m, 700mX700m, 800mx800m, 900mX900m and 1000mX1000m area, respectively.

Fig. 4 shows the performance of the 7 schemes under different network scales with the same node density. DD802 performs almost the same as in the experiment of varying node density in a fixed area, with a throughput of 105 packets on average and 100% data delivery ratio. Obviously, neither node density nor network scale influences the performance of DD802.

DDSMAC has an overall decreasing performance as the network scale increases. Since a large network scale implies that source nodes are on average more hops away from the sink node, packets from a source node may experience worse synchronization, longer delay and higher dropping probability on the routing path. Therefore, as the network scale increases, the best duty cycle for DDSMAC in terms of throughput increases from 20% for 600mX600m area to 40% for 1000mX1000m area, due to the fact that larger duty cycles always have better data delivery ratios.

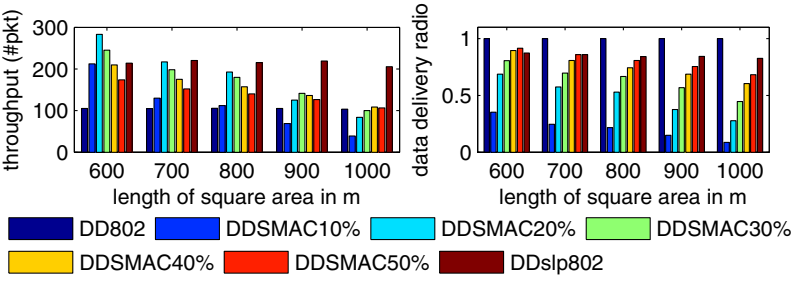


Fig. 4. Performance comparison under varying network scale

DDslp802, on the other hand, has a steady performance in both throughput and data delivery ratio. As fixed node density guarantees that a sensor has on average a fixed number of neighbors, enlarging the network scale does not increase the routing redundancy. Meanwhile, IEEE 802.11 has reliable data delivery, hence DDslp802 does not suffer from extended multi-hop transmission.

Varying Number of Sources. In this experiment, the number of source nodes varies from 2 to 26. Fig. 5 shows the performance of the 7 schemes. DD802, as usual, has constant performance.

Overall, DDSMAC has an obvious improvement in throughput when the number of source nodes increases from 2 to 5, but the improvement slows down when the number of source nodes continues to increase. As the number of source nodes increases, the sink node can on average reach one of the source nodes in fewer hops. The fewer hops a transmission experiences, the higher data delivery ratio it will have, see Fig. 4. However, as the number of source nodes increases, the probability that the sink node has at least one neighboring source node improves from fast to slowly (more than one 1-hop source node does not help DDSMAC since DDSMAC does not utilize routing redundancy). Accordingly, the improvement in throughput slows down.

DDslp802 increases in throughput, but decreases in data delivery ratio. This is because source nodes are the second redundancy that sleep Directed Diffusion could use besides redundant routing nodes. Since Directed Diffusion reinforces

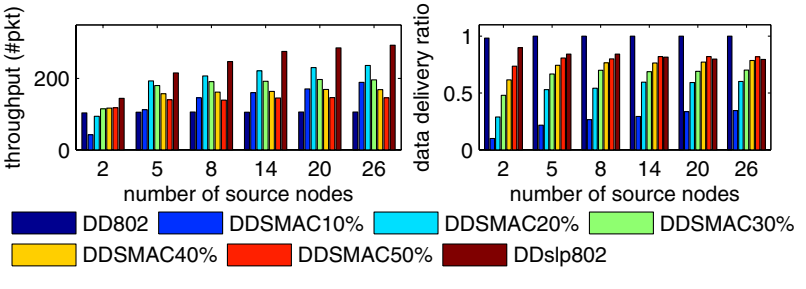


Fig. 5. Performance comparison under varying number of source nodes

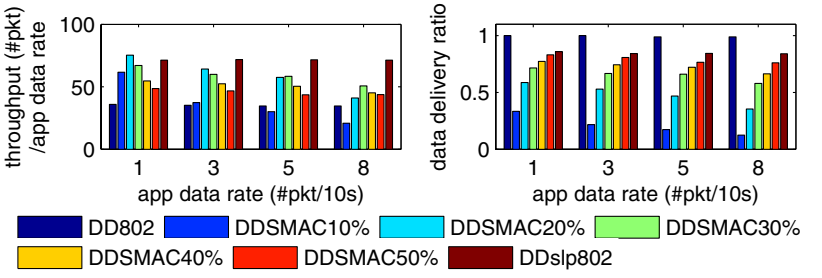


Fig. 6. Performance comparison under varying application data rate

one path (one source node) at once to deliver data, other redundant source nodes are put to sleep for later use. Hence, once a source node dies, another source node can be reinforced to continue the data delivery. However, the more source nodes to reinforce, the more chances DDslp802 will incur a QoS pause. Consequently, the data delivery ratio decreases slightly as the number of source nodes increases.

Varying Data Rate. In this experiment, the application data rate at each reinforced source node varies from 1 packet per 10s to 8 packets per 10s. Fig. 6 shows the throughput of the 7 schemes scaled by the application data rate and their corresponding data delivery ratio. The throughput of DD802 increases proportionally to the application data rate, while the data delivery ratio is 100%.

In absolute terms, DDSMAC also receives more packets as the application data rate increases, but relatively, the throughput of DDSMAC is decreasing if scaled by the application data rate. The decreasing performance of DDSMAC can also be seen in its data delivery ratio. This is because SMAC cannot provide reliable data delivery, as shown in Section 4.1, due to the contention in the network. Higher application data rates mean more packets to send, and hence more contention and packet loss in a fixed period of time. As the application data rate increases, the best duty cycle in terms of throughput increases from 20% for 1 packet per 10s to 30% for 8 packets per 10s. Larger duty cycles can alleviate contention in the network, as sensors are sleeping for a shorter time.

DDslp802, on the other hand, has throughput proportional to the application data rate and maintains the same data delivery ratio as the application data rate changes. As IEEE 802.11 provides reliable data delivery, the data delivery ratio of DDslp802 drops only because of the QoS pauses introduced by sleeping Directed Diffusion. When the node density and the number of sources are kept the same, the redundancy that DDslp802 can utilize does not change. Hence, DDslp802 reinforces on average the same number of redundant paths, and consequently has the same chances of introducing a QoS pause.

4.3 Performance of Sleeping at Both Layers

In this experiment, we compare the performance of sleeping schemes at both routing and MAC layers, with and without cross-layer coordination, as described

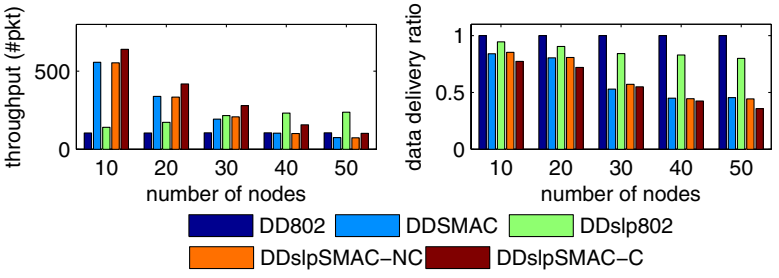


Fig. 7. Performance of multi-layer sleeping with/without cross-layer coordination

in Section 3.3, as we vary the node density. For simplicity, we mention the combination of sleeping Directed Diffusion and SMAC with its best duty cycle in terms of throughput under DDSMAC as DDslpSMAC, followed by -NC or -C for the case of without coordination or with coordination, respectively.

Fig. 7 shows the throughput and the data delivery ratio of 5 schemes. As we can see, DDslpSMAC-NC has very similar performance in both throughput and data delivery ratio as DDSMAC even though sensors can now sleep at both layers. Since no coordination is implemented between the routing layer and the MAC layer, sleeping Directed Diffusion does not have higher priority than SMAC to schedule the sensors. When a sensor is turned off by sleeping Directed Diffusion, it will still be woken up periodically by SMAC according to its duty cycle. Hence, sensors are mostly following SMAC to sleep and wake up.

On the other hand, with certain coordination, DDslpSMAC-C significantly improves the throughput compared with DDSMAC, since it not only reduces idle listening during data delivery but also utilizes network redundancy. When the node density is not very high, DDslpSMAC-C performs the best in terms of throughput among all the sleeping schemes. However, when the node density increases, DDslpSMAC-C cannot achieve as high throughput as DDslp802, since the severe contention at the MAC layer greatly impairs the most vulnerable but most important data delivery at the routing layer - unicasting positive reinforcement packets and unicasting high rate DATA packets. The loss of positive reinforcement packets delays the path establishment, while the loss of DATA packets impairs the throughput directly. Hence, DDslpSMAC-C cannot further improve the throughput compared with DDslp802, although sensors sleep at both layers. DDslpSMAC has a data delivery ratio lower than the data delivery ratio of either DDSMAC or DDslp802, because both QoS pauses and SMAC unreliability are degrading the performance. Meanwhile, DDslpSMAC has similar standard deviations with DDSMAC in either throughput or data delivery ratio, while DDslp802 has smaller standard deviations (not shown in the figure).

These preliminary results show the benefit of employing cross-layer coordination between the routing and MAC layers to achieve higher throughput under low network density. Hence, a smart decision can be made by the power

management of wireless sensor networks to dynamically choose the best sleeping scheme (single layer sleeping or multi-layer sleeping with coordination) as the network density changes over time. We expect in our future work that sleeping at both routing and MAC layers with cross-layer coordination can also outperform single layer sleeping schemes when varying network scale, the number of source nodes and the application data rate, as long as the contention in the network is low. Moreover, we believe the performance of DDslpSMAC-C can be further improved and may outperform single layer sleeping schemes all the time as more sophisticated cross-layer coordination is employed.

5 Conclusions

In this paper, we analyze sleeping schemes conducted by routing protocols and MAC protocols individually and simultaneously, for wireless sensor networks, in order to determine the best method for sleeping under different network and application scenarios. While conclusions are made by simulating networks that run routing protocols with/without sleeping (Directed Diffusion and our newly proposed sleeping Directed Diffusion) and MAC protocols with/without sleeping (IEEE 802.11 and SMAC), the conclusions can also be applied to other routing protocols that turn off sensors when they are not involved in the data delivery, and other MAC protocols that use duty cycles to save energy. In general, our results show that routing layer sleeping is more suitable for networks with high redundancy or high contention, while MAC layer sleeping is more sensitive to contention, and hence is a good choice for light traffic applications under small scale networks. Furthermore, we show that cross-layer coordination can significantly improve the network throughput under low contention scenarios, when routing layer sleeping and MAC layer sleeping are employed simultaneously. Therefore, a smart decision can be made by a power manager to dynamically switch to a sleeping scheme at the routing layer or the MAC layer or both layers with cross-layer coordination, as the network conditions or application requirements change over time. Moreover, more sophisticated cross-layer coordination has the potential to further improve the network throughput and outperform single layer sleeping schemes in all cases.

Acknowledgments. This work was supported in part by the National Science Foundation under grant # CNS-0448046 and in part by a Young Investigator grant from the Office of Naval Research, # N00014-05-1-0626.

References

1. Zoghi, M.R., Kahaei, M.H.: Sensor Selection for Target Tracking in WSN Using Modified INS Algorithm. In: 3rd Internatioanl Conference on Information and Communication Technologies: From Theory to Applications, pp. 1–6 (2008)
2. Xu, Y., Heidemann, J., Estrin, D.: Geography-informed Energy Conservation for Ad Hoc Routing. In: 7th Annual International Conference on Mobile Computing and Networking, pp. 70–84 (2001)

3. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wirleess Networks. Wireless Networks 8(5), 481–494 (2002)
4. Zheng, R., Kravets, R.: On-demand Power Management for Ad Hoc Networks. In: 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, pp. 481–491 (2003)
5. Wang, H., Wang, W., Peng, D., Sharif, H.: A Route-oriented Sleep Approach in Wireless Sensor Network. In: 10th IEEE Singapore International Conference on Communication systems, pp. 1–5 (2006)
6. Ye, W., Heidemann, J., Estrin, D.: Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. IEEE/ACM Trans. on Networking 3, 493–506 (2004)
7. Polastre, J., Hill, J., Culler, D.: Versatile Low Power Media Access for Wireless Sensor Networks. In: 2nd International Conference on Embedded Networked Sensor Systems, pp. 95–107 (2004)
8. Kulik, J., Rabiner, W., Balakrishnan, H.: Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In: 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 174–185 (1999)
9. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J.: Directed Diffusion for Wireless Sensor Networking. IEEE/ACM Trans. on Networking 11(1), 2–16 (2003)
10. Chang, J., Tassiulas, L.: Maximum Lifetime Routing in Wireless Sensor Networks. IEEE/ACM Trans. on Networking 12(4), 609–619 (2004)
11. Perillo, M., Heinzelman, W.: DAPR: A Protocol for Wireless Sensor Networks Utilizing an Application-based Routing Cost. In: Wireless Communications and Networking Conference, vol. 3, pp. 1540–1545 (2004)
12. Luo, H., Luo, J., Liu, Y., Das, S.K.: Adaptive Data Fusion for Energy Efficient Routing in Wireless Sensor Networks. IEEE Trans. on Computers 55(10), 1286–1299 (2006)
13. http://circuit.ucsd.edu/curts/courses/ECE284_F05/lectures/ECE284_F05_L07_EnergyEfficiency_2pp.pdf
14. http://www.isi.edu/nsnam/ns/

Distributed Task Synchronization in Wireless Sensor Networks

Marc Aoun[1], Julien Catalano[2], and Peter van der Stok[1]

[1] Philips Research, High Tech Campus 34, 5656AE Eindhoven, The Netherlands
{marc.aoun,peter.van.der.stok}@philips.com
[2] Enensys Technologies, Le Germanium, 80 avenue des Buttes de Coesmes, 35700 Rennes, France
julien.catalano@enensys.com

Abstract. A WSN is envisaged to consist of a large number of nodes, each combining sensing, processing and wireless communication capabilities. The nodes have to cooperate to achieve a global function required by the user. Towards achieving this goal, multiple tasks are performed on all nodes or a subset of nodes in the network. The subject of this paper is the synchronized execution of periodic tasks at different nodes, for example: sensing tasks or a periodic on/off switching of radio transceivers. The method is based on synchronizing timer interrupt occurrences at different nodes to a single reference, leading to tight synchronization of tasks over nodes. Measurements show that task synchronization is achieved on our hardware platform with an average time difference of 5 μs between the starts of temporally related tasks on different nodes.

Keywords: task synchronization, time synchronization, real-time operating system, timestamping, synchronized sampling, clock drift.

1 Introduction

Major technological advances in the field of MicroElectroMechanical Systems (MEMS) have allowed the production of small size, low cost nodes with sensing, processing, and wireless communication capabilities. Ad-hoc networks made of these nodes are referred to as Wireless Sensor Networks (WSNs). Although initially investigated for military applications, the usage of WSNs is currently envisioned for a multitude of civilian application fields such as agriculture, security, habitat monitoring, and preventive machine maintenance. One of the main application fields is the healthcare domain, with what is referred to as Body Sensor Networks (BSNs) that are used to monitor the health of patients by sensing their vital signs and other relevant aspects such as the level of activity.

Nodes in a WSN operate autonomously, and have to cooperate to achieve a main objective required by the application. Each node has to perform a number of tasks, where some tasks are periodic, some are event-driven and some are performed only once. In this paper, we focus on periodic tasks. A set of identical

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 150–165, 2009.
© Springer-Verlag Berlin Heidelberg 2009

tasks, common to all nodes or at least a subset of nodes, are periodically executed by different nodes. This distributed execution is usually not automatically synchronized among nodes.

To synchronize the measurements performed by the tasks, a synchronized execution of the tasks has been developed. Thanks to this work, synchronizing the sensing tasks over a set of nodes has resulted in samples that are taken within an interval of a few μs, therefore providing the high accuracy needed for data correlation. Examples where this is needed are:

1. Studying gait and limb orientation of a subject, for either improving sport performance or reeducation of stroke patients. A simultaneous measurement of the orientation in space of two or more parts of the body is required.
2. Synchronized on/off switching of radio transceivers for power saving in Time Division Multiple Access (TDMA) schemes. Synchronized tasks on the sender and receiver nodes switch their radio modules at the same time for communication and then turn them off to reduce power consumption.

Task synchronization complements time synchronization. With task synchronization, measurements or actions can be as close as 5 μs; without task synchronization but with time synchronization, measurements or actions can be several milliseconds apart.

With the exception of the work done by Werner-Allen et al. [1], the topic of distributed task synchronization in WSNs has gained limited attention so far. Previous work focused on providing a WSN with time synchronization, and therefore on providing the nodes with the same global time that is then used to timestamp sampled data. However, there is no guarantee that the sampled data are measured close enough in time. Timestamps help to correlate data, and figure out when a certain sensing sample was made, but they do not allow the samples to be actually made within a very narrow interval.

Going beyond time synchronization, our work aims at providing a WSN with a task synchronization service. We describe in this paper a method for achieving this purpose. The observation that tasks are started by an operating system (OS) when the interrupts of a specific timer occurs, and knowing that the timers on the nodes are started at different moments, lead to the insight of synchronizing network-wide the interrupt occurrences of this specific OS timer. Synchronizing the operating system timer ticks leads in a straightforward to task synchronization. In what follows, we refer to the synchronization of the OS timer interrupt occurrences as "Tick Interrupt Alignment" (TIA).

We tested the devised method on our target platform that runs FreeRTOS, an open source Real-Time Operating System [8]. Measurements show that the method achieves task synchronization in a single-hop network with a time offset of 5 μs on average between the start of execution of the temporally related tasks on different nodes. In the remaining parts of this paper, we will discuss some of the related work, present the theoretical foundations of our method, the practical implementation on our target platform, and finally, the measurements that we obtained on real nodes.

2 Time Synchronization Versus Task Synchronization

We deem it important at this early stage of the paper to clarify the difference between time synchronization and task synchronization, and dissipate any confusion that might arise because of their similar nomenclature. Time synchronization per se provides a way to translate the local time of a node to a global time that becomes common to all the nodes in the network. On the other hand, task synchronization aims at executing related tasks at different nodes in synchrony with each other. It *might*, towards that end, use information provided by a running time synchronization service.

One might be tempted at this point to present a "simple" solution to the task synchronization problem; it would seem straightforward to first define when a task should be executed on the global timescale. The defined global time could then be translated into a corresponding local time value. The node could wait until this local time value is reached, and then trigger the execution of the task. As much as such a solution seems attractive at first glance, it suffers from a major flaw; it totally overlooks the fact that in a multitasking OS environment, task execution is dictated by and can only be triggered in harmony with a specific OS timer. Even when the method claims to trigger the task, the actual execution will get delayed until the OS timer fires. Since the OS timer runs at its own pace, differently from node to node, this method's performance would be significantly limited.

3 Related Work

Previous work has mostly dealt with the concept of time synchronization. Some of the proposed time synchronization protocols such as TPSN [3] periodically compensate the accumulated time offset between two nodes (instantaneous synchronization), whereas other protocols, such as RBS [2] and FTSP [4], go beyond a simple instantaneous synchronization by estimating the relative clock offset evolution (first order time derivative) between the clocks of the nodes. This approach reduces the frequency with which time synchronization packets should be sent and therefore reduces the energy consumption needed for synchronization while keeping the same accuracy. A precise timestamping at the MAC layer during the transmission and reception of time synchronization messages is proposed in [3], [4], and [5] to minimize the jitter in timestamping delay.

The issue of coordinating task execution, which is the specific topic of our paper, is addressed by Werner-Allen et al. [1]. The authors use the word synchronicity to refer to a coordinated "firing" of pulses by the sensor nodes. The idea is based on a work done by R. Mirollo and S. Strogatz [7] that provides a model (M&S model) for synchronous firing of biological oscillators. In the M&S model, a node has an internal time t, that starts at 0 and is incremented until reaching the period value. Once this value is reached, the node fires and t is reset to 0. When another node notices that its neighbor fired, it adjusts its time t forward in order to reduce its time to fire. The adjustment is determined by a

"firing function" f and a constant ϵ smaller than 1. Mirollo and Strogatz prove that a set of sensor nodes in an all-to-all topology will achieve synchronicity in firing if f is smooth, monotonically increasing and concave down [1]. Later work done by Lucarelli and Wang [6] proved that synchronicity would still be achieved in a multi-hop network where nodes can only listen to a subset of nodes in the network.

The authors in [1] base their work on the M&S model, while taking into account the characteristics of wireless communication such as message delays, message loss and medium access control based on CSMA schemes. A simple approximation of the function f is used to overcome the limited floating-point arithmetic capabilities of the nodes. The main presented contribution is a delaying of the time adjustments by one time cycle to avoid problems that might arise due to message delays and out-of-order messages reception. The authors report a synchronicity with an accuracy up to 100 μs. The two main shortcomings of the method are the long setup time and the high rate of message exchange needed to achieve and maintain synchronicity.

4 Tick Interrupt Alignment (TIA)

4.1 General Overview

A tick interrupt is a periodic timer interrupt event that happens once the operating system timer counter in the node reaches a predefined end value E and is reset back to its starting value S. Operating systems such as FreeRTOS [8] use tick interrupts as a reference to start executing a scheduled task. Once a tick interrupt happens, the task picked by the operating system scheduler will start being executed.

Tick interrupts happen at all nodes in the network. It is highly unlikely that tick interrupts occur at the same moment at all nodes or that the number of tick interrupts contained in a time frame $[t_1 : t_2]$ on a reference timescale is the same at all nodes. The reason is the difference in start-up times of the nodes, which translates itself into different start-up times of the counters. Additionally, even if timers are started at the same moment,the deviation of the real oscillator frequency from the nominal frequency provokes a drift of the individual start times with respect to each other.

The main objective of Tick Interrupt Alignment (TIA) is to achieve and maintain a network-wide synchronized occurrence of tick interrupts, such that the tick interrupt becomes a synchronized event over the network that is then used as reference for executing the temporally related tasks in a synchronous fashion on different nodes. Such an approach alleviates the coordination effort needed in centralized coordination schemes where a single entity directs all the other nodes on the moment the task should be executed.

TIA bears some similarities to the firefly-inspired synchronicity [1] discussed in Sect. 3. Both approaches aim at synchronizing the occurrence of an event over a set of nodes (e.g. the timer loop that periodically generates a firing event). On the other hand, major differences between the two methods also exist; our

method synchronizes the timer interrupt events of all nodes in a network to that of a single master whereas in [1], nodes receive firing events information from all neighboring nodes, and adjust their firing schedules based on this information, without relying on a reference node. In addition, TIA does not make use of a firing function f nor of a constant ϵ to define adjustments that should be made, and therefore, needs less time to converge.

TIA is based on a periodic exchange of counter values coupled with an estimate of the evolution of the periods T of the timer counter with respect to the master's. By taking the period's evolution into account, TIA is able to reduce the rate of instantaneous offset corrections, maintaining task synchronization at no additional message exchange cost in between these correction interventions. This is in great contrast to the firefly-inspired method [1] where the period evolution is not taken into account, which requires a node to send a message every time its event fires.

In [1], no proposal is made for operating system support to start synchronized tasks. As already mentioned, in operating systems such as FreeRTOS, a task execution cannot be planned to occur at any point in time. Tasks can be scheduled to start executing only at the boundaries of timer loops, defined by tick interrupt occurrences. Ignoring the scheduling characteristics significantly limits the performance of any task synchronization method.

4.2 Timer Functionality

Sensor node controllers are usually equipped with timers. A timer counts from a start value S to an end value E, set to default values S_{default} and E_{default} respectively. We will refer to a counting cycle from S to E followed by a reset back to S as "timer round". The default number of unitary counting steps in a timer round is given by

$$\tau_{\text{round,default}} = |E_{\text{default}} - S_{\text{default}}| + 1 \,. \tag{1}$$

Counting is based on the oscillations of a crystal oscillator. Two counting directions are possible: incrementing counting where $S_{\text{default}} < E_{\text{default}}$, and decrementing counting where $S_{\text{default}} > E_{\text{default}}$. In general, S_{default} is equal to 0 for incrementing counters and E_{default} is equal to 0 for decrementing counters. Incrementing counting and decrementing counting are illustrated in Fig. 1.

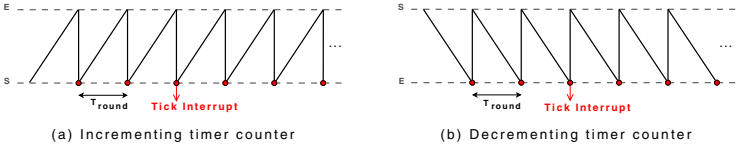


Fig. 1. Incrementing (a) and decrementing (b) timer counter

Once a timer round is completed, a timer interrupt is signaled. The timer interrupts on node i occur periodically, with a period $T_{\text{round},i}$ given by

$$T_{\text{round},i} = \frac{\tau_{\text{round,default}}}{f_{\text{nominal}}} \ . \tag{2}$$

Based on the functionality provided by the timer, node i can calculate its local time $t_{\text{local},i}$ using

$$t_{\text{local},i} = n \times T_{\text{round},i} + \left| \frac{V_i - S_{\text{default}}}{f_{\text{nominal}}} \right| , \tag{3}$$

where:

$$\begin{cases} n & = \text{number of completed timer rounds,} \\ V_i & = \text{current value of the timer counter,} \\ f_{\text{nominal}} & = \text{nominal frequency of the oscillator.} \end{cases}$$

Without loss of generality, the subsequent explanation of Tick Interrupt Alignment will be based on an incrementing time counter.

Counting from S to E starts when the node is powered on. Since nodes are in general not turned on at exactly the same moment, counting is not synchronized on all nodes. This is illustrated in Fig. 2, for two nodes M and N.

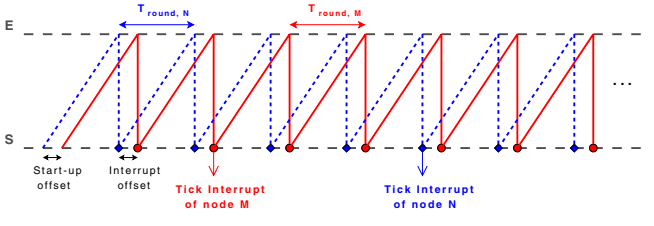


Fig. 2. Timer interrupts not synchronized due to start-up time offset.

Timer counters are based on a crystal oscillator, and variations in the oscillator frequency are very likely to exist between oscillators present on different nodes. Furthermore, the frequency of an oscillator can vary with time, causing a variation in the timer rounds' period within the same node. The difference in oscillator frequency between M and N implies that even in the absence of a start-up time offset, the nodes are not likely to exhibit synchronized timer interrupt occurrences. This is illustrated in Fig. 3.

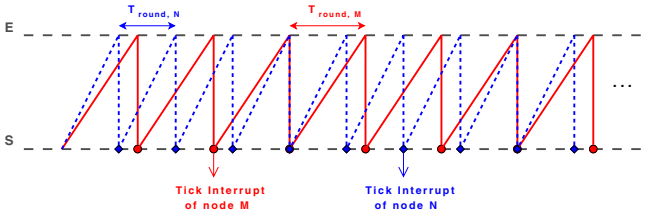


Fig. 3. Timer interrupts not synchronized due to differences in oscillator frequencies

Due to the initial time offset and the difference in oscillator frequencies, local times of different nodes will seldom be equal. The relationship between the local time $t_{\text{local,M}}$ of node M and the local time $t_{\text{local,N}}$ of node N is

$$t_{\text{local,M}} = (1 + \delta)t_{\text{local,N}} + \beta \ , \tag{4}$$

where:

$$\begin{cases} \delta = \text{relative clock skew evolution between M and N,} \\ \beta = \text{initial time offset between M and N.} \end{cases}$$

The job of a time synchronization service is limited to providing estimates of δ and β and as such, approximating the relationship defined in (4). The job of our task synchronization service is equalizing the values of V_i and $T_{\text{round},i}$ to align the tick interrupts.

4.3 Aligning the Timer Interrupts

In this section, tick interrupts of node N are synchronized to those of node M. We define a timer counter snapshot to be a value reached by the timer counter at a given moment of interest. This value is bounded by the start and end value of the timer round during which the snapshot is made.

When the relative clock skew evolution δ between M and N is equal to 0, interrupt alignment can be achieved with a single exchange of snapshots followed by a one-time modification of the end value of a timer round at node N to correct for the counting offset. The snapshots, one from node M, and another from node N, are made at the occurrence of an event common to both nodes. We will refer to the snapshots taken at M and N by snp_{M} and snp_{N} respectively, and to the difference between them by D, given by

$$D = snp_{\text{M}} - snp_{\text{N}} \ . \tag{5}$$

In order for D to precisely reflect the counting offset that exists between the two timer counters, the common event should *ideally* happen simultaneously on the two nodes, i.e. without indeterministic delay components. In practice, this can be approximated by sending a broadcast message to nodes M and N, which reaches

the nodes M and N almost simultaneously, followed by almost equal interrupt service routine (ISR) activation delays at both nodes.

Once D becomes available at node N, the latter can modify the end value of one of it's timer rounds to achieve interrupt alignment. Two options are available at this point: As Soon As Possible (ASAP) Alignment, and Delayed Alignment. In ASAP Alignment, the end value is set to E_{ASAP}. In Delayed Alignment, the end value is set to E_{Delayed}. These values are calculated using

$$E_{\mathrm{ASAP}} = E_{\mathrm{default}} - D \,, \tag{6}$$

$$E_{\mathrm{Delayed}} = E_{\mathrm{default}} + (E_{\mathrm{default}} - D) \,. \tag{7}$$

ASAP Alignment achieves tick interrupt synchronization one timer interrupt earlier than Delayed Alignment, hence the nomenclature. The reasons to use one or the other is determined by the value of $(E_{\mathrm{ASAP}} - D)$ and the delay until its ISR is executed, as outlined below.

The end value of a timer round should be modified in the tick ISR. When a timer interrupt is signaled, the timer counter is reset to S and counting continues, regardless of whether or not the ISR has been executed or its execution is delayed until the processor is free.

When E_{ASAP} is smaller than the delay that exists between the occurrence of a timer interrupt and the execution of its corresponding ISR, it can happen that the end value is set to E_{ASAP} only *after* the timer counter has already exceeded this value. In such a case, the timer counter will continue counting until it reaches the maximum value that its register can hold. It will then be reset back to 0, and count till it reaches E_{ASAP}. Only then would a timer interrupt occur again. This behavior is unwanted. On the other hand, in Delayed Alignment, the correction of the end value is delayed by one complete timer round, therefore avoiding any problems due to delays in interrupt processing. For decrementing timer counters, the problem does not exist since it is the start value that will be modified in the ISR.

A decision should thus be made in the ISR on whether to use an ASAP Alignment or a Delayed Alignment, based on the calculated E_{ASAP} and an estimate of the worst-case delay that can exist between a timer interrupt and the execution of its ISR.

For the case where δ is equal to 0, once the offset is corrected for, all subsequent timer interrupts at node N will occur synchronously to the interrupts at node M.

Since δ is in reality unlikely to be equal to 0, a one-time correction would not be enough to keep the tick interrupts synchronized. A periodic offset correction can be performed. The period with which corrections are made depends on the value of δ and on the required synchronization precision. Such a periodic instantaneous correction approach performs poorly in terms of energy consumption since it requires a high rate of message exchange to keep the counting offset below a required limit.

A different approach, proposed and adopted in this work, is based on a combination of periodic instantaneous corrections, referred to as Major Correction Steps (MCS), and a correction phase that spans multiple consecutive timer

rounds and during which the skew evolution is compensated for. During this phase, referred to as Skew Compensation Phase (SCP), no message exchange is needed. Compensating for the skew evolution on counting reduces the need for instantaneous corrections. This reduces the rate of message exchange and thus energy consumption.

In general, SCP takes place directly after each MCS, and will keep on being applied until the next MCS is executed. Therefore, a SCP covers a period of time bounded by two MCSs. The reason why an MCS is performed periodically is that the skew evolution estimate is periodically updated. Each time a new skew evolution estimate is available, a MCS is applied to correct for any small offset that might have accumulated during the SCP. Each MCS serves as a starting point for a new SCP that uses the most recent available skew evolution estimate.

In SCP, node N uses the estimate of its clock skew evolution relative to M to deduce an end value E_δ that gives the same effective timer period as that of M. E_δ will be used as end value for all timer rounds until the next MCS. It is given by

$$E_\delta = \frac{\tau_{\text{round,default}}}{(1+\delta)} + S_{\text{default}} - 1 \; . \tag{8}$$

Whereas (8) would likely result in non-integer values, the end value of the counter can only be an integer. Therefore, in the SCP, the end value is set to the integer that is closest to the result of (8). The fractional offset between this result and the integer value is accumulated. A correction is made once the accumulated value becomes bigger or equal to 1 or any predefined integer value.

Accuracy of the MCS suffers from a δ not equal to 0; (6) and (7) would not provide accurate results. There is usually a delay between the moment the snapshots are made and the MCS is performed. A non-zero skew evolution and this delay will result in outdated snapshot information for the MCS.

To perform snapshot updates and scale conversions, the following procedure is performed: When snp_{N} is made, the number n_1 of completed timer rounds is recorded. In the ISR during which MCS is performed, a new snapshot $snp_{\text{N,new}}$ is made and n_2, the new number of completed timer rounds, is recorded. The total number of counting steps $\tau_{\text{total,N}}$ elapsed between snp_{N} and $snp_{\text{N,new}}$ is found. for the first MCS and any MCS that is not preceded by a SCP this number satisfies

$$\begin{aligned} \tau_{\text{total,N}} =& (n_2 - n_1 - 1)\tau_{\text{round,default}} + snp_{\text{N,new}} \\ &+ (E_{\text{default}} - snp_{\text{N}} + 1) \; . \end{aligned} \tag{9}$$

For an MCS that is preceded by a SCP, $\tau_{\text{total,N}}$ is found using

$$\begin{aligned} \tau_{\text{total,N}} =& (n_2 - n_1 - 1)\frac{\tau_{\text{round,default}}}{1+\delta} + snp_{\text{N,new}} \\ &+ (E_\delta - snp_{\text{N}} + 1) \; . \end{aligned} \tag{10}$$

$\tau_{\text{total,N}}$ is then used to estimate the corresponding number of elapsed counting steps $\tau_{\text{total,M}}$ at node M, such that

$$\tau_{\text{total,M}} = (1+\delta)\tau_{\text{total,N}} \; . \tag{11}$$

An updated snapshot value $snp_{\text{M,new}}$ is found using

$$snp_{\text{M,new}} = [\tau_{\text{total,N}} - (E_{\text{default}} - snp_{\text{M}} + 1)] \\ \text{mod}(\tau_{\text{round,default}}) \; . \tag{12}$$

To perform the MCS, the end value of N's timer counter is set to either E_{ASAP} or E_{Delayed} given by

$$E_{\text{ASAP}} = \frac{E_{\text{default}}}{1+\delta} - \left(\frac{snp_{\text{M,new}}}{1+\delta} - snp_{\text{N,new}} \right) \\ - \frac{\delta}{1+\delta} \tag{13}$$

and

$$E_{\text{Delayed}} = E_{\text{ASAP}} + E_{\delta} \; . \tag{14}$$

Snapshot updates and snapshot scale conversions may produce a too large overhead for the tick ISR and lead to extra delays in the system. In practice, they can be omitted at the expense of less accuracy.

5 Coexistence of TIA and Time Synchronization

A time synchronization algorithm uses timestamps of the local time and the global time, to find an estimate of the skew evolution and the offset between them. As mentioned in sect 4.2, local time is measured by keeping track of the progress of a timer.

Our TIA algorithm uses the skew estimate provided by the time synchronization service to complete its SCP. The evolution estimate indirectly reflects the functioning of the oscillator used by a timer. As such, to provide the skew evolution estimate relevant for TIA, the time synchronization should either be based on the OS timer used by TIA, or another one that shares the same oscillator as the OS timer. In our case, it is based on the OS timer.

Time synchronization requires a free-running timer. If the counting progress of the timer is "artificially" perturbed by changing the start or end values of the timer rounds, the estimation mechanism would result in completely erroneous estimates. In both its phases (MCS and SCP), the TIA algorithm introduces modifications to the free-running timer, perturbing its normal progress. Therefore, to achieve a coexistence between TIA and time synchronization, the perception of a free-running counter is maintained by reconstructing the raw timer values.

6 Tick to Task Mapping

With synchronized ticks, task synchronization is achieved when at every node the same tick interrupt is chosen to activate the task.

A potential method is: the nodes one hop from a given master can receive information about the next execution time of the task at the master node, and then provide similar information to the nodes at the next hop, and so forth.

The alternative that we chose to implement is based on node-local decisions concerning task execution times. The method is based on using the global time to deduce the number of completed timer rounds at the master, and start the periodic task when this number becomes a multiple of the task's period. In such a way, even though each node achieves TIA asynchronously from other nodes and starts executing the task regardless of whether or not other nodes are ready (i.e. achieved TIA), the execution moments will become aligned at some point due to the fact that they are multiples of the same value. Multiple periodic tasks can be synchronized independently based on the achieved tick interrupt alignment.

7 Implementation and Practical Aspects

Task synchronization software have been developed for our node hardware. In addition, a time synchronization service has been implemented. Measurements were executed on this hardware and software. All results are real-life measurements and no simulations.

7.1 The Target Platform

Our testbed consists of a single-hop network of 5 SAND nodes. Each SAND node incorporates, among other modules, a CoolFlux DSP and a CC2420 radio chip. The CoolFlux is a 24-bit dual-Harvard architecture Digital Signal Processor (DSP). It features two 64 Kwords data memories (24 bits) and one 64 Kwords program memory (32 bits) [12].

The CC2420 operates in the frequency range 2400–2483 MHz, divided into 16 channels. It has a data rate of 250 kbps and an output power ranging from -25 to 0 dBm [10].

The CoolFlux DSP in our platform runs a real-time operating system, FreeRTOS [8], and an implementation of the IEEE 802.15.4 standard [9].

One node in the network acts as time master. The remaining nodes perform time synchronization and task synchronization relative to the master node.

7.2 FreeRTOS

FreeRTOS [8] is an open source Real-Time Operating System. It provides the choice between preemptive and cooperative scheduling policies. In this test we use the preemptive scheduling mode. FreeRTOS's scheduler is a round robin priority-based scheduler. Each task has a priority level. The total number of priority levels is defined by the program designer. A timer is used to periodically

generate interrupts that define when tasks start executing. Each time such a *tick interrupt* occurs, the scheduler checks which tasks are in the ready state and picks the one that has the highest priority for execution. Tasks having the same priority level share the processor in a round-robin fashion by allowing a task to run for only one time slice (time between two tick interrupts).

A chosen task starts to get executed until it blocks, or its time slice ends. The next equal-priority task in line will then start to be executed directly after. Once no more tasks of the highest priority level are ready, the scheduler moves to the next lower priority level, and so forth.

The period $T_{\text{round,DSP}}$ with which tick interrupts are generated is configurable. This period defines the time it takes for a timer round from start to end value to be completed. By default, in FreeRTOS, $T_{\text{round,DSP}}$ is equal to 1000 μs.

7.3 Time Synchronization

Time synchronization provides us with an estimate of the relative clock skew evolution δ. The nodes are running a one-hop version [5] of FTSP [4], with MAC-layer timestamping and skew evolution estimation. The SFD interrupt specified in [9] has been used to timestamp time synchronization messages at both the sender side (master) and the receiver side. The SFD interrupt occurs at the sender side when the SFD byte of the IEEE 802.15.4 frame (time synchronization message) has been sent, and at the receiver side when this same byte has been received. Neglecting propagation time, this interrupt occurs almost simultaneously at both the sender and receiver [13]. The CC2420 datasheet [10] mentions a small delay of approximately 2 μs because of bandwidth limitations in both the transmitter and the receiver. The average value of this delay is corrected for.

Every P seconds, a message sent by the master results in a pair of timestamps, (master Ts, node Ts) at each node. This pair is added to the linear regression table. Linear regression is performed on a set of r timestamp pairs.

From a computation efficiency point of view, we use 24-bit variables to hold time values on the DSP. One consequence of our choice is that the overflow value of the local time on the DSP is equal to 16777216 μs. We slightly modified

Table 1. Time synchronization results (mean (μ), standard deviation (σ), maximum and minimum) using Linear Regression (LR) for different values of r and P

		$P = 2$ sec	4 sec	6 sec	8 sec			$P = 2$ sec
2-points LR	min (μs)	-6	-7	-7	-9	6-points LR	min (μs)	-3
	max (μs)	6	8	6	8		max (μs)	4
	μ (μs)	1.40	1.48	1.6	2.0		μ (μs)	0.87
	σ (μs)	1.14	1.25	1.33	1.64		σ (μs)	0.79
4-points LR	min (μs)	-4	-5			8-points LR	min (μs)	-3
	max (μs)	4	6				max (μs)	3
	μ (μs)	0.91	0.89				μ (μs)	0.86
	σ (μs)	0.82	0.91				σ (μs)	0.72

$T_{\text{round,DSP}}$ to a power of 2 value, 1024 μs, to have an integer number of timer rounds for each overflow cycle of the local time.

The small time overflow value introduces constraints on the maximum possible value that the product $r \times P$ can take; $r \times P$ should be smaller than the overflow period. Therefore, with a simple 2-points linear regression ($r = 2$), P can at most be equal to 8 seconds. Table 1 shows the time synchronization performance between the master node and the other nodes in our testbed.

Based on this implementation, our performance compares to the time synchronization performance reported in [4] and [5]. Our results show that combining MAC-layer timestamping and skew evolution estimation results in accurate time synchronization, with an achievable average absolute error in the sub-microsecond range.

7.4 Tick Interrupt Alignment Practical Implementation

The TIA implementation follows the analytical model presented in Sect. 4.3. The SFD interrupt outlined in Sect. 7.3 was used as reference event for generating snapshot pairs $(snp_{\text{M}}, snp_{\text{N}})$. To reduce the message overhead further, the snapshots were included in the same messages exchanged for the time synchronization service. The snapshot pairs are used to perform the MCS. The skew evolution estimate relative to the master is used to perform the SCP. The time synchronization period was set to two seconds and a simple 2-points LR was used. This means that a new skew evolution estimate is delivered by the time synchronization service every two seconds, triggering a new MCS. To alleviate computation requirements, the processing delay and the skew evolution were assumed to be negligible when performing the MCS, an assumption that greatly simplifies the calculations (no snapshot updates and scale conversion required anymore) without incurring substantial errors.

7.5 Task Synchronization

Before TIA is achieved, the "to-be-synchronized" periodic task will remain blocked waiting on a semaphore. Once TIA is completed, the semaphore is released. The next time the task is accessed for execution, a task preamble is executed only once. This part finds an estimate of the current global time and deduces the number n_{master} of completed timer rounds at the master node using

$$n_{\text{master}} = \left\lfloor \frac{t_{\text{global}}}{T_{\text{round,DSP}}} \right\rfloor . \tag{15}$$

This number is consistent over all nodes. The preamble ends with a call to a delay function that delays the execution of the task by a number Δn of local timer rounds, such that

$$\Delta n = P_{\text{task}} - (n_{\text{master}} \mod(P_{\text{task}})) . \tag{16}$$

After the delay expiry and this initial execution, all of the following executions of the task will be performed each P_{task}.

7.6 Testing Procedure and Results

To test the performance of our method, a signal level on a GPIO pin of each DSP was changed each time a synchronized task, with a period equal to 512 tick interrupts, starts to execute. The signals were visualized on an oscilloscope. Fig. 4 shows an example, illustrating the signals of the master and a second node.

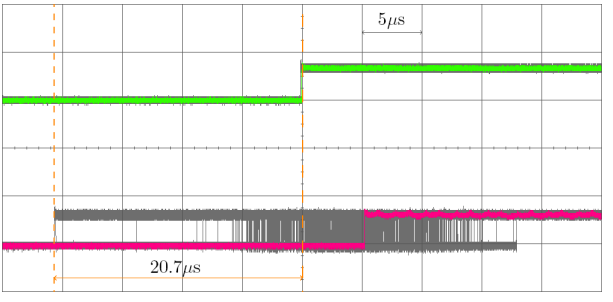


Fig. 4. Task execution start-up on each node is represented by a signal level change from low to high. The oscilloscope's "persist" function is used here to keep track of the task synchronization offset.

Of interest is the value of the absolute offset between each node and the master. This value was determined at each node for 1000 task executions, thus resulting in a total of 4000 offset values. The cumulative distribution function and statistics of the offset are presented in Fig. 5. The average absolute offset is equal to 5.05 μs. The minimum offset value is equal to 0 μs and the maximum absolute offset value is equal to 20.7 μs. The results show that task synchronization based

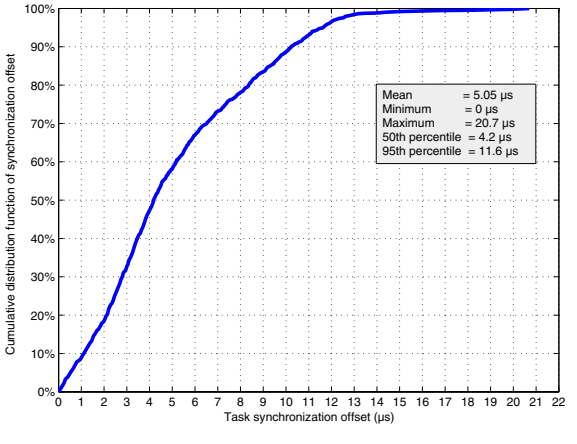


Fig. 5. Cumulative distribution function of the task synchronization absolute offset values

on Tick Interrupt Alignment can achieve a high degree of task synchronization accuracy, with 90% of the offset values being smaller than 10 μs.

We identify the following potential reasons why the average of 5.05 μs achieved by task synchronization is larger than the time synchronization accuracy of 1.4 μs (with r = 2, P = 2 seconds):

1. Existence of jitter in the operating system context switch on each DSP.
2. Existence (in practice) of delay between making snapshots at the master and listening nodes.
3. Snapshot updates and scale conversions are not implemented.
4. Errors in the skew evolution estimate (would contribute to an accumulated error in the SCP).

8 Conclusion and Future Work

We presented in this paper a method that is capable of achieving and maintaining a synchronized network-wide execution of periodic tasks in a Wireless Sensor Network. The method is based on providing synchronized timer interrupt occurrences that are used by the operating system as reference to launch tasks. The test results show that our method is capable of achieving a high level of task synchronization accuracy, with an average time offset of 5 μs between the start of execution of the same task on different nodes in a single-hop scenario. In the near future, we intend to investigate the performance of our method in a multi-hop environment. Preliminary tests on multi-hop time synchronization showed an increase of less than 0.3 μs per hop. It can be stated at this point that task synchronization performance will not substantially degrade when increasing the number of hops and that it will exhibit a performance similar to multi-hop time synchronization. Nevertheless, only the actual testing that we intend to perform would confirm our expectations.

Acknowledgments

We are thankful to Prof. Dr. Petri Mähönen, Head of the Department of Wireless Networks at RWTH Aachen (Germany), for making this work possible. This work is partially financed by the European Commission under the Framework 6 IST Project "Wirelessly Accessible Sensor Populations (WASP)" (IST-034963).

References

1. Werner-Allen, G., Tewari, G., Patel, A., Nagpal, R., Welsh, M.: Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In: 3rd ACM Conference on Embedded Networked Sensor Systems (Sensys 2005), pp. 142–153 (2005)
2. Elson, J., Girod, L., Estrin, D.: Fine-Grained Network Time Synchronization Using Reference Broadcasts. In: 5th Symposium on Operating System Design and Implementation (OSDI 2002), pp. 147–163 (2002)

3. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync Protocol for Sensor Networks. In: 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003), pp. 138–149 (2003)
4. Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The Flooding Time Synchronization Protocol. In: 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 39–49 (2004)
5. Aoun, M., Schoofs, A., van der Stok, P.: Efficient Time Synchronization for Wireless Sensor Networks in an Industrial Setting. In: 6th ACM Conference on Embedded Networked Sensor Systems (Sensys 2008), pp. 419–420 (2008)
6. Lucarelli, D., Wang, I.-J.: Decentralized Synchronization Protocols with Nearest Neighbor Communication. In: 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 62–68 (2004)
7. Mirollo, R.E., Strogatz, S.H.: Synchronization of Pulse-Coupled Biological Oscillators. SIAM Journal on Applied Mathematics 50(6), 1645–1662 (1990)
8. FreeRTOS^{TM}Homepage (accessed, August 2008), http://www.freertos.org/
9. IEEE Std 802.15.4-2006: IEEE Standard for Information Technology – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) (2006)
10. Chipcon (Texas Instruments): CC2420 data sheet. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver (2006)
11. Atmel Corporation: ATMega128L data sheet. 8-bit Microcontroller with 128K Bytes In-System Programmable Flash (2006)
12. Roeven, H., Coninx, J., Ade, M.: CoolFlux DSP: The Embedded Ultra Low Power C-programmable DSP Core. In: Proceedings of the International Signal Processing Conference (GSPx), Santa Clara (2004)
13. Cox, D., Jovanov, E., Milenkovic, A.: Time Synchronization for ZigBee Networks. In: 37th Southeastern Symposium on System Theory (SSST 2005), pp. 135–138 (2005)

Solving the Wake-Up Scattering Problem Optimally

Luigi Palopoli[1], Roberto Passerone[1], Amy L. Murphy[2], Gian Pietro Picco[1], and Alessandro Giusti[3]

[1] Dip. di Ingegneria e Scienza dell'Informazione (DISI), University of Trento, Italy
{luigi.palopoli,roberto.passerone,gianpietro.picco}@unitn.it
[2] Fondazione Bruno Kessler—IRST, Trento, Italy
murphy@fbk.eu
[3] Dept. of Electronics and Information, Politecnico di Milano, Italy
giusti@elet.polimi.it

Abstract. In their EWSN'07 paper [1], Giusti et al. proposed a decentralized *wake-up scattering* algorithm for temporally spreading the intervals in which the nodes of a wireless sensor network (WSN) are active, and showed that the resulting schedules significantly improve over the commonly-used random ones, e.g., by providing greater area coverage at less energy costs. However, an open question remained about whether further improvements are possible. Here, we complete the work in [1] by providing a (centralized) *optimal* solution that constitutes a theoretical upper bound for wake-up scattering protocols. Simulation results shows that the decentralized algorithm proposed in [1] comes within 4% to 11% of the optimum. Moreover, we show that the modeling framework we use to derive the solution, based on integer programming techniques, allows for a particularly efficient solution. The latter result discloses important opportunities for the practical utilization of the model. The model is also general enough to encompass alternative formulations of the problem.

1 Introduction

The operation of a wireless sensor network (WSN) node is usually characterized by (long) periods of inactivity spent in a low-power stand-by state, interleaved with (short) periods where the expected sensing, computation, and communication duties are carried out. This duty-cycling is used to minimize energy consumption, therefore maximizing the lifetime of the network at large. Nevertheless, the performance of WSN applications critically depends on the quality of the duty-cycling schedule, ensuring that the right nodes are active at the right time. A random schedule may lead to an inefficient use of resources. Consider Figure 1, where three nodes cover, with overlaps, a given area. Ensuring that nodes 2 and 3 are *not* simultaneously active saves energy. Indeed, since a point in the target area may be monitored by multiple nodes, it is possible to switch some off as long as the remaining active nodes cover the target area.

In their EWSN'07 paper [1], Giusti et al. presented a decentralized protocol that leverages this observation by *scattering* the nodes' wake-up times, therefore

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 166–182, 2009.
© Springer-Verlag Berlin Heidelberg 2009

taking advantage of the overlap among the nodes' sensing range. They demonstrated the effectiveness and practical relevance of their *wake-up scattering* protocol in several common WSN scenarios, including the coverage problem above. However, although they showed that their protocol yields significant improvements over random schedules, they did not answer the question about whether further improvements can be obtained and, if so, how significant.

This paper provides an answer to this question, by giving a means to compute the *optimal* schedule of wake-up times that maximizes the area covered by a set of sensor nodes. Our technique, based on integer linear programming (ILP), is inherently centralized as it assumes global topology knowledge. However, the significant theoretical contribution of this paper is that, under the reasonable assumption of a constant and periodically-repeated awake interval, the resulting optimization problem can be radically simplified, enabling an efficient solution. The significant computational speed-up we gain allows us to deal with a large number of nodes in a reasonable time. Therefore, besides providing a theoretical upper bound against which to evaluate distributed solutions, our technique can be used as an effective design tool in cases where parameters such as the deployment topology are under control. Moreover, we envision a combination of the two techniques, where our optimal off-line solution determines the initial configuration of the system at deployment time, and the distributed algorithm in [1] is used online to adapt to topology changes.

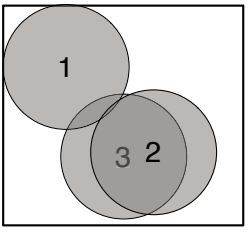


Fig. 1. A topology where wake-up scattering matters

Section 2 presents a complete description of the problem, followed by the description of our approach and a proof that our formulation, which offers significant computational speed-ups, provides results equivalent to the general case. In Section 3 we compare quantitatively our optimal solution against the original decentralized one in [1]. The evaluation confirms that the latter algorithm, albeit very simple, is very efficient, as it comes within 4% to 11% of the optimum. Moreover, through our mathematical formulation we are able to gain insights into the relationships among coverage, lifetime, and node density.

Our modeling framework is actually general enough to represent problems other than coverage, similarly affected by duty-cycling: we discuss these issues in Section 4. Finally, after a concise survey of related approaches in Section 5, we end the paper with our concluding remarks in Section 6.

2 System Model and Solution Algorithm

Our approach assumes[1] a *target area* A to be monitored using a set of *sensor nodes* $\mathcal{N}$. For each node $n \in \mathcal{N}$, we know its *position* in the target area A,

[1] For simplicity, we describe the problem in two dimensions: the extension to a three-dimensional space is straightforward.

represented as a pair of coordinates (x_n, y_n), as well as its *sensing range* n_A, i.e., the area that is directly monitored by n. We assume that the sensitivity of a node in its sensing range is constant, and that the boundaries of the sensing range are sharp. In other words, the complete topology of the problem is known in advance. An example was already shown in the introduction in Figure 1.

The system operates according to a periodic schedule. The period, called the *epoch*, is denoted by E. In our problem, the lifetime of the system is fixed and determined by the *awake* or *activation interval* of a node, i.e., the interval during which a node is awake in the epoch. We assume that the awake interval is the same across all nodes, and that each node is woken up only once per epoch. Alternative formulations with different awake intervals or multiple activations per epoch are possible at the expense of increased computational complexity.

Our objective is to maximize the coverage by scheduling the nodes' wake-up time. At any time t, the set of nodes awake at t defines a covered area $S(t)$, equal to the union[2] of the sensing areas of all awake nodes. Our objective is thus to maximize the integral of $S(t)$ over the epoch. Clearly, constraining the awake interval and the epoch is tantamount to fixing the network lifetime, and one could argue that we address the problem of maximizing the coverage *given* the lifetime. However, as shown in Section 4, solving the problem for different values of the lifetime allows us to explore the tradeoffs between lifetime and coverage.

Our method to solve the optimization problem goes through two steps, which are described in detail in Section 2.1 and 2.2, respectively.We first consider and solve the *spatial partitioning* problem, i.e., how to determine a *finite* number of regions in the target area that, without loss of accuracy, can be used to model the system topology and the area overlaps. Then, we set up an integer linear program expressing the scheduling problem, with coverage constraints for each node and region found in the previous step. The optimization problem is then solved using standard linear programming and branch-and-bound techniques, and yields the optimal schedule for the given objective function and constraints.

2.1 Spatial Partitioning

The optimization problem can be setup as a linear program by enforcing a certain level of coverage on the target area A, using appropriate constraints discussed in Section 2.2. This approach is feasible if A is discretized into a finite number of regions, to avoid generating constraints for each of the infinitely many points in A. Here we use the concept of *field* [2], i.e., regions of A that are invariant relative to node coverage. In other words, any two points within one such region are covered by exactly the same nodes. In this case, it is sufficient to consider only one point per region, or, equivalently, consider the region as a whole.

Several algorithms have been proposed for field computation. Slijepcevic and Potkonjak approximate the computation by a regular sampling of the area [2], while Tian and Georganas rely on geometric approximations to compute the required intersections [3]. Huang and Tseng propose an exact and efficient algorithm for deciding if an area is covered by at least k sensors by considering only

[2] Unlike the *sum* of these areas, overlaps are not counted multiple times.

the perimeter of the sensing area [4] and reasoning on the angles of intersection. This way, the computational complexity is reduced to $O(nd \log n)$, where n is the number of nodes and d is the average number of nodes that overlap a given node. Here, we present a simpler algorithm, with the same complexity, for the case of rectangular rather than the more traditional circular sensing areas, and use coordinates instead of angles. Our algorithm works without change when the sensing area is represented by *unions of rectangles*, including the case of unconnected sensing areas. This feature can be used to approximate arbitrary shapes, albeit with increased computational complexity. In any case, our optimization is *independent* of the shape of the sensing area and the way regions are computed.

In the following sections, we will use $r : A \to 2^{\mathcal{N}}$ to denote the function that for each point $p \in A$ returns the set of nodes that cover p. A region (or field) ρ is the largest subset of points of A which are covered by the same set of nodes, We denote the set of regions by the symbol $\mathcal{R}$. For each region $\rho \in \mathcal{R}$, the corresponding area is returned by the function $w : \mathcal{R} \to \mathbb{R}_+$. Given that there is a one-to-one mapping, we use $r(\rho)$ to denote the set of nodes covering region ρ. Note also that each region need not necessarily be connected, even when the sensing areas of the nodes are connected and convex. Thus, this kind of spatial partitioning cannot be obtained using a simple regular discretization.

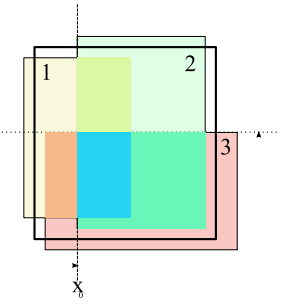


Fig. 2. Target area and scanning sequence

The partitioning algorithm works by scanning the target area horizontally, left to right, stopping at every vertical boundary of a node's range, as shown in Figure 2. At every stop, the algorithm performs a vertical scan, from bottom to top, that computes the regions found at that horizontal position, and updates their area. This is sufficient, since the computation of the optimal schedule does not require the shape of the regions, but only their area and corresponding set of covering nodes. The latter conveniently identifies the region, as discussed above. For example, in Figure 2, the vertical scan at x_0 finds a sequence of regions corresponding to the sets of nodes

Region	Nodes	Area
ρ_0	$\emptyset$	56
ρ_4	{ 1, 3 }	96
ρ_1	{ 1 }	144
ρ_5	{ 2 }	244
ρ_2	{ 1, 2 }	140
ρ_6	{ 2, 3 }	272
ρ_3	{ 1, 2, 3 }	160
ρ_7	{ 3 }	112

Fig. 3. Regions for the topology of Figure 2

$$\{3\}, \{2,3\}, \{1,2,3\}, \{1,2\}, \{2\}. \tag{1}$$

Areas are accumulated incrementally, and computed up to the next stop in the scanning sequence. The procedure is shown in detail in Algorithm 1. To perform the horizontal scan, we first build a sequence *h_scan* of all the vertical boundaries of the nodes, ordered by their horizontal position (line 2.1). We then loop through this sequence (line 2.1) and stop at every element. To perform the vertical scan we also build a sequence, denoted *v_scan*, by inserting the horizontal boundaries of

Algorithm 1. Spatial partitioning algorithm.

```
1.  h_scan = list of vertical boundaries of nodes, ordered by their horizontal position
2.  v_scan = ∅
3.  R = ∅
4.  for i = h_scan.begin() to h_scan.end() do
5.    if i is left boundary of node n then
6.      Insert the horizontal boundary of n in v_scan ordered by vertical position
7.    else // i is the right boundary of node n
8.      Erase the horizontal boundary of n from v_scan
9.    end if
10.   h_extent = horizontal extent to next stop
11.   node_set = ∅
12.   for j = v_scan.begin() to v_scan.end() do
13.     if j is bottom boundary of node n then
14.       Insert n in node_set
15.     else // j is the top boundary of node n
16.       Remove n from node_set
17.     end if
18.     v_extent = vertical extent to next stop
19.     area = h_extent · v_extent
20.     R = R ∪ {node_set}
21.     node_set.area = node_set.area + area
22.   end for
23. end for
```

the nodes that intersect the scan at the current horizontal position in the order of their vertical position. For efficiency, this sequence is constructed incrementally: it is initialized to empty (line 2.1) and at every step of the horizontal scan we insert or remove the horizontal boundaries of the node at that position, according to whether we are entering (from the left) or exiting (to the right) the sensing area of the node (lines 2.1 and 2.1). In the example of Figure 2, the vertical scan at x_0 would already have the ordered entries $(3_{bot}, 1_{bot}, 3_{top}, 1_{top})$. At x_0, since we are entering node 2, we add 2_{bot} and 2_{top} in the correct order, to obtain

$$(3_{bot}, 2_{bot}, 1_{bot}, 3_{top}, 1_{top}, 2_{top}). \tag{2}$$

The vertical scan starts with the loop at line 2.1. During this phase, we keep track of the set of nodes *node_set* for which the current point is in range. This set is updated at every step of the scan according to whether we are entering the node's range from the bottom (line 2.1) or exiting it from the top (line 2.1). For example, by scanning sequence (2) we obtain the regions of sequence (1). After computing the area of the *node_set*, we add it to the set $\mathcal{R}$ (line 2.1). If the region was already present in the set, we do not add a new element but simply update its area (line 2.1). The result of applying Algorithm 1 to the topology of Figure 2 is shown in Figure 3, where each region ρ is associated with its covering nodes and its area. Note that the area covered outside the target area A is ignored.

2.2 Computation of the Optimal Schedule

As discussed, we consider periodic schedules of a fixed duration E, called the *epoch*. We assume that every node wakes up exactly once per epoch, and operates for a defined time I, the *awake interval*. The optimization problem consists of finding the time within the epoch at which each node should wake up (the

wake-up time) in order to maximize the total coverage. Intuitively, this can be achieved by scheduling the operation of overlapping nodes at different times, while nodes that do not overlap could be scheduled concurrently. The original decentralized algorithm described in [1] approximates this idea by scattering the awake times of neighboring nodes, which are more likely to overlap. Finding the optimal solution to this problem is complex, and a naive formulation easily results in algorithms that are impractical even for small networks. Nonetheless, it is possible to simplify the problem by taking advantage of the following result.

Theorem 1. *Let the duration E of the epoch be an integer multiple of the awake interval I. Then there exists a schedule such that every node wakes up at some integer multiple of I (within the epoch) which maximizes the average coverage.*

Instead of considering arbitrary wake-up times distributed on the real line, we can discretize the problem by dividing the epoch E into $L = E/I$ equal *slots* of length I, where every node is awake in exactly one slot. Thus, to solve our problem, we do not need to look at schedules in which nodes overlap only partially in time, thereby pruning a large portion of the solution space. Assuming that the epoch is an integer multiple of the awake interval may seem a severe limitation. However, high power savings can be achieved only at low duty-cycles, where our approach has sufficient granularity to provide accurate trade-offs.

To prove the theorem (see [5] for an extended proof), we assume that the epoch E is initially finely discretized into s slots, $s \gg L$, and the awake interval spans d slots (i.e., $I = d \cdot \frac{E}{s}$). Since, by hypothesis, E is an integer multiple of I, we have $s = L \cdot d$. The continuous case can be obtained when s tends to infinity. Figure 4 depicts our notation. Slots are ordered and identified by their position $0 \leq k \leq s-1$. In addition, all operations on slot indices are done modulo s. Node n has a wake-up time $0 \leq w_n \leq s-1$. We say that a node is *aligned* if it is scheduled at an integer multiple of the awake interval, i.e., if $w_n \bmod d = 0$. We say that a *schedule* is aligned if *all nodes* in the schedule are aligned. The line that identifies the slots at which aligned nodes can be scheduled is called an *alignment boundary*. The slots between two consecutive alignment boundaries is called an *alignment region*. If the schedule is aligned, then all nodes are scheduled at the alignment boundaries, that is, they pairwise either completely overlap in time, or they do not overlap at all. Given an arbitrary schedule, we can partition the set $\mathcal{N}$ of nodes into the set $\mathcal{A}$ of those that are already aligned, and the set $\mathcal{B}$ of those which are not. For every slot k, we denote by $\mathcal{A}_k$ the set of aligned nodes that are awake at slot k, and by $\mathcal{B}_k$ the set of non-aligned nodes that are awake at slot k. Because all non-aligned nodes span across an alignment boundary, the set of non-aligned nodes awake at the slots across an alignment boundary are the same. Thus, for $k \bmod d = 0$, $\mathcal{B}_{k-1} = \mathcal{B}_k$.

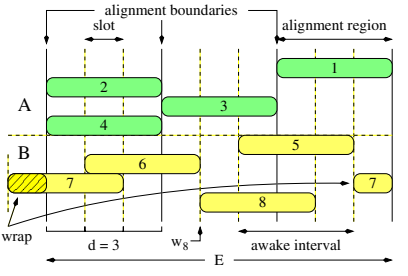


Fig. 4. Node alignment

Similarly, the aligned nodes in slots that belong to the same alignment region are, of course, the same. Therefore, $\mathcal{A}_k = \mathcal{A}_{k'}$ for $dm \leq k, k' \leq d(m+1) - 1$.

We compute the gain (positive or negative) in covered area that is obtained by shifting the schedule of *all* the *non-aligned* nodes together by one slot to the left or to the right. To do so, we must compute the coverage before and after the shift. The change in coverage depends on the area overlaps before and after: without overlaps all schedules are equivalent. More precisely, because we shift all non-aligned nodes together, the *gain* g_k^+ of slot k for a right shift is given by the area overlap between the non-aligned and the aligned nodes before the shift, minus the area overlap of the same non-aligned nodes with the aligned nodes in the new slot, after the shift. Formally, let $a_k \subseteq \mathbb{R}^2$ be the region covered by the nodes in $\mathcal{A}_k$, $b_k \subseteq \mathbb{R}^2$ the region covered by the nodes in $\mathcal{B}_k$, and $A : \mathbb{R}^2 \to \mathbb{R}$ be the function that to a subset of $\mathbb{R}^2$ gives the corresponding area. Then,

$$g_k^+ = A(a_k \cap b_k) - A(a_{k+1} \cap b_k). \tag{3}$$

A slot k which is not near an alignment boundary, however, gives no gain, since in that case $a_k = a_{k+1}$. Recalling that $L = s/d$, and by similar arguments for left shifts, we can therefore express the total gains as

$$G^+ = \sum_{i=0}^{L-1} g_{di-1}^+, \qquad G^- = \sum_{i=0}^{L-1} g_{di}^-. \tag{4}$$

The key observation is that a right and a left shift give gains that are equal, but of opposite sign. In fact, let $di = k$ be a slot marking the beginning of an alignment region. As observed before, $b_{k-1} = b_k$. Therefore,

$$\begin{aligned} g_{di}^- &= g_k^- = A(a_k \cap b_k) - A(a_{k-1} \cap b_k) \\ &= A(a_k \cap b_{k-1}) - A(a_{k-1} \cap b_{k-1}) = -g_{k-1}^+ = -g_{di-1}^+ \end{aligned}$$

By matching corresponding terms in G^+ and G^-, it follows that $G^+ = -G^-$.

If a shift of the non-aligned nodes does not result in any new node being aligned, a further shift of the same non-aligned nodes in the same direction will give the exact same area gain. For instance, let G_1^+ be an initial right shift. Under the new configuration, if the aligned node do not change, it must be $G_1^+ = -G_2^-$. By the same argument, $-G_2^- = G_2^+$, and therefore $G_1^+ = G_2^+$. The same holds for a left shift that does not alter the partition.

The proof of Theorem 1 proceeds by induction. Given a non-aligned schedule, it is easy to construct an aligned schedule which has equal or better coverage. It is sufficient to shift the non-aligned nodes in the direction of zero or positive gain (since gains are opposite, they are either both zero or one of them is positive). Once a new node is aligned, we proceed by shifting the remaining non-aligned nodes, until the schedule is fully aligned. Since all moves had zero or positive gain, the new schedule has equal or better coverage. Thus, given an optimal schedule, we are always able to find an aligned schedule with the same coverage.

While Theorem 1 guarantees the existence of an aligned optimal schedule, it does not tell us how to find an optimal schedule in the first place. However, it

ensures that the coarsest possible discretization of the problem is also optimal, which allows us to set up a significantly simpler optimization problem. We set up a boolean linear program (BLP), which can be solved by standard commercial and open source tools. We compute the coverage starting from the partition of the target area in regions by introducing a set of binary *coverage* variables $C_{\rho,k}$ which take value 1 whenever region ρ is covered during slot $k \in [0, L-1]$, and 0 otherwise. Since regions are by definition disjoint, using the coverage variables $C_{\rho,k}$ and the area function A, the covered area can be computed as

$$S = \sum_{k=0}^{L-1} \sum_{\rho \in \mathcal{R}} C_{k,\rho} \cdot A(\rho). \tag{5}$$

Our objective is to maximize the total area S.

The value of $C_{\rho,k}$ depends on the schedule. To model it, we introduce a set of binary *scheduling* variables $x_{n,k}$ which take the value 1 whenever node n is awake in slot k, and 0 otherwise. Then, we express the relation between the coverage and the scheduling variables as linear optimization constraints. This relation can be easily understood by observing that $C_{\rho,k} = 1$ if and only if at least one of the nodes that cover ρ is active in slot k, i.e., when $x_{n,k} = 1$ for some $n \in r(\rho)$. Likewise, $C_{\rho,k} = 0$ whenever all of the nodes that cover ρ are inactive in slot k, i.e., when $x_{n,k} = 0$ for all $n \in r(\rho)$. This translates into the following constraints:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1],\ C_{\rho,k} \leq \sum_{n \in r(\rho)} x_{n,k} \tag{6}$$

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1], \forall n \in r(\rho),\ C_{\rho,k} \geq x_{n,k} \tag{7}$$

$$\forall n \in \mathcal{N}, \sum_{k=0}^{L-1} x_{n,k} \leq 1 \tag{8}$$

Constraint (6) forces the condition $C_{\rho,k} = 0$ when no node covers a region in a slot, (7) ensures that $C_{\rho,k} = 1$ when at least one node covers a region, and finally (8) ensures that a node wakes up at most once per awake interval.

The optimization problem can be further simplified by observing that the coverage variables $C_{\rho,k}$ are constrained by Equations (6) and (7) to never take values strictly between 0 and 1, regardless of whether they are defined as integer or real variables. Thus, we can relax $C_{\rho,k}$ to a continuous variable, which is more efficiently handled by optimization algorithms, by adding the constraints:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1],\ 0 \leq C_{\rho,k} \leq 1 \tag{9}$$

More complex situations can also be handled, however at the expense of increased computational complexity. We describe some of these in Section 4, and compare them against related work in Section 5.

3 Optimal vs. Distributed: Evaluation

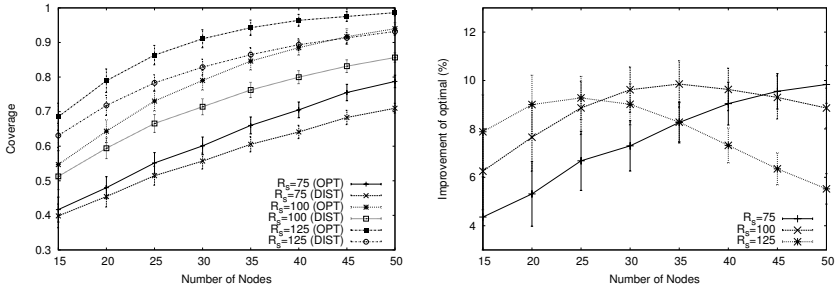
As stated previously, our motivation for this work was to find an optimal solution for the wake-up scattering problem, and determine how close the original distributed solution [1] comes to it. This section presents an evaluation by using covered area as the primary performance metric. Specifically, if $S(k)$ is the area covered during slot k, the covered area during an epoch is $\frac{1}{L}\sum_{k=0}^{L-1} S(k)$. Intuitively, this calculates the largest coverage possible given a network topology, ignoring regions of the field that are not covered by any sensors. Our evaluation uses GLPK [6], an open source linear programming kit[3], to compute the optimal schedule, and the simulator described in [1] to compute the distributed ones.

While the full details of the distributed solution are available in [1], for completeness we briefly describe its key functionality here. The distributed solution starts by assigning each node a random wake-up time. Through a simple sequence of message exchanges, each node learns which of its neighboring nodes wakes up immediately before and after it in the epoch. It also learns when these nodes wake up relative to its own wake-up time, then selects its new wake-up time to be approximately in the middle of these two points in time. This process repeats at all nodes until no significant changes in the wake-up time are made at a single step. Although the resulting schedule is not aligned on slot boundaries, by Theorem 1 it can be slotted without loss of coverage.

The majority of our experiments were performed in a 500×500 area. The number of nodes varied from 15 to 50, allowing us to consider increasing node densities. To achieve statistical significance, each point in our plots represents an average over 100 distinct topologies. Additionally, because the distributed scattering protocol does not change the wake-up order among nodes and is therefore affected by the initial random wake-up configuration, the results average 10 different initializations for each topology. This is sufficient as the variation with different initializations is small. We considered square sensing areas, the sensing range R_S being half of the edge length. For the distributed solution, we used a circular communication range $R_C = R_S\sqrt{2}$, creating a sensing square exactly inscribed in the communication circle. The resulting node densities for $R_S = 100$ range from 2.7 to 9.3. To verify that the results carry over to larger topologies, we also ran experiments with similar densities obtained with up to 200 nodes in a 1000×1000 area, as reported at the end of this section.

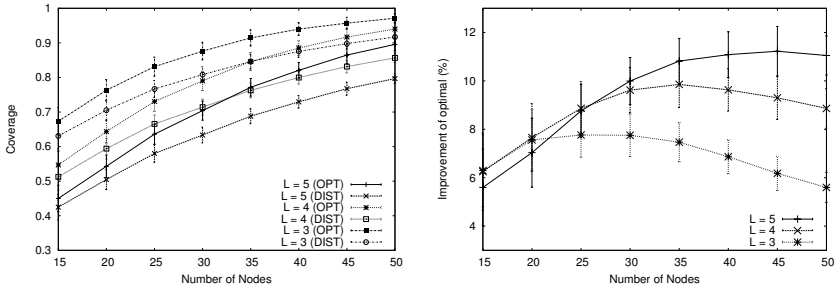
We studied two main configurations, corresponding to the two main dimensions of the problem. In the first setting, we fixed the number of slots per epoch and therefore the duty cycle, and varied the sensing range. In the second, we reversed roles, fixing the sensing range and varying the number of slots. In each case, a node is awake for only one slot. The results in Figures 5(a) and 6(a) clearly show that as either the sensing range or duty cycle increase, both solutions cover larger areas. The standard deviation is a few percent points. In

[3] Although the GLPK documentation mentions a 10-100 performance gap w.r.t. commercial tools, our computation times on a common workstation were in the order of a few hours even for a very complex topology with a hundreds of nodes.



(a) Absolute coverage (b) Percent improvement of optimal

Fig. 5. Fixed number of slots ($L = 4$), variable sensing range



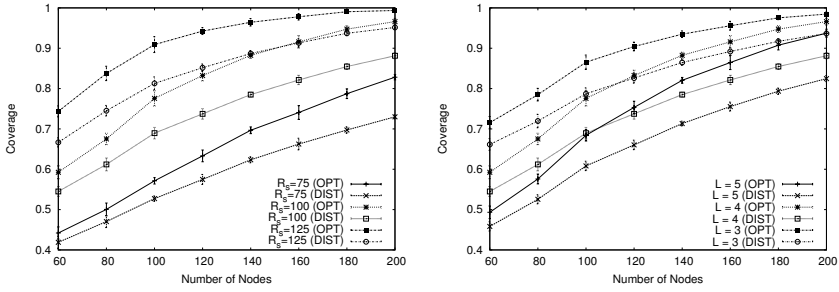
(a) Absolute coverage (b) Percent improvement of optimal

Fig. 6. Fixed sensing range ($R_S = 100$), variable number of slots

general, the evaluation shows how the distributed wake-up scattering in [1], albeit very simple, is remarkably effective and yields schedules performing within 4% to 11% of the optimal ones computed with the technique presented here.

Interesting trends emerge when studying the percentage improvement of the optimal solution w.r.t. the distributed one, defined as $\frac{optimal - distributed}{optimal}$ and reported in Figures 5(b) and 6(b). In both plots, each line, representing either a different sensing range or number of slots, shows a general trend where the percent improvement increases to a certain point, then begins to decline. This indicates that at low and high densities the optimal and distributed solutions behave similarly, while at intermediate densities the optimal one performs better.

The initial similarity of the two solutions is due the fact that at low densities very few nodes overlap in space. In a sense, at low densities the system is far away from saturation and both solutions find schedules in which overlapping sensing areas overlap very little in time. As the density increases, overlaps become inevitable and the system must scatter the wake up times to eliminate them as much as possible. Unfortunately, to properly manage spatial overlaps, node locations must be considered and nodes close to one



(a) Fixed number of slots ($L = 4$) (b) Fixed sensing range ($R_S = 100$)

Fig. 7. Experiments with 1000×1000 area

another should be more scattered. However, because the distributed solution does not consider the distance between nodes and does not change the sequence of node wake-up times, with more dense scenarios it is more likely that physically close nodes are next to one another in the wake-up sequence, and therefore less scattered. At low densities, physically close nodes are also close in the schedule, however, with fewer total neighbors, the interval between wake-up times is larger. In contrast, the optimal solution considers neither connectivity constraints nor initial wake-up times, and therefore does not suffer from the same problem.

The tail of the curves, where the two solutions again perform similarly, is due to the saturation of the covered area in either time or space. With a fixed number of slots, saturation is due to the increased number of nodes covering a single point in space. Instead, with a fixed sensing range, saturation arises from the higher number of nodes awake at a given time. In both cases, the distributed solution again approaches the optimal.

Because saturation depends on the sensing range and number of slots, the peak of each line appears at different network densities. In Figure 5(b), larger sensing ranges saturate faster, while in Figure 6(b), in scenarios with fewer slots, when nodes are awake for longer, saturation occurs earlier. Interestingly, in Figure 5(b) the sensing range $R_S = 75$ does not reach saturation, which is however likely to appear at higher densities.

Finally, to demonstrate that our results scale to larger topologies, we considered a 1000×1000 area containing from 60 to 200 nodes. This yields the same density as in the previous experiments. Because the computation of the optimal schedules in this case took two days to complete, we ran only 10 topologies for each point. The trends in Figure 7 clearly follow those in Figures 5(a) and 6(a), suggesting that similar trends hold for large topologies as well. This is expected as both solutions work by considering overlapping sensing ranges, which by nature are localized. Put another way, a 1000×1000 topology behaves similarly to a scenario where four of the smaller 500×500 topologies are juxtaposed to form

a 1000×1000 square area. The main difference is that, in the larger topologies, the *edges* of the four quadrants of the area are properly scattered, while in a solution which considers each quadrant independently, the edge nodes would not be properly scattered.

4 Using and Extending the Model

Although our original motivation was to compare the optimal and distributed solutions, we explore here two additional research contributions: some practical applications of the optimal results and an extension to a new class of problems.

As shown in Section 3, the distributed solution closely approximates the optimal solution, coming within 4% to 11% in all cases. Therefore, the optimal solution can be naturally applied as a pre-deployment tool to select system parameters that meet the application needs. For example, consider a system with given lifetime and coverage constraints. Plots generated by the offline optimization tool, such as the one in Figure 8 showing lifetime as the ratio of the epoch length and the slot length, clearly show the trade-off between lifetime and coverage and thus the system behavior with various settings. Although the distributed solution will not achieve equivalent results, such an evaluation still provides the developer with valuable insight into system behavior prior to deployment. Alternate topologies may also be considered, either fixing the locations of some nodes or changing the number of nodes. Because these simulations are relatively straightforward to perform, extensive pre-deployment evaluation can be performed at little cost. In case of a controlled deployment, not only the centralized algorithm provide a quick evaluation of the results that can be achieved, but it can be also used as a full-fledged design tool to decide the initial schedule of nodes. In this case, the distributed algorithm can be used to adapt to changes (e.g., depletion of the battery on some nodes) affecting the system topology.

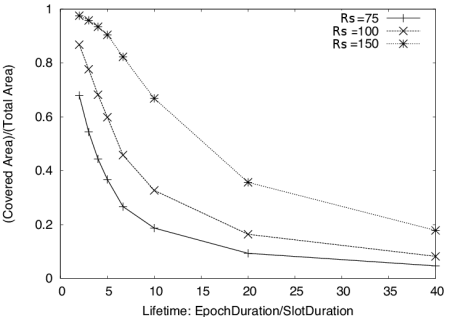


Fig. 8. Lifetime vs. covered area with $L = 4$ slots

While the above provide an immediate, practical use for the optimal solution, we also explored how the formulation proposed in Section 2.2 can be applied to a wider class of problems other than coverage. For example, we can handle the inverse problem: given a required coverage, find a schedule that *maximizes the lifetime* of the network. If we take the common definition of lifetime based on failure of the first node, the goal can be achieved by minimizing the maximum energy consumed by any node in a single epoch. We can also relax several of the assumption made previously, yielding a more flexible solution space and perhaps

additional efficiency. For example, we can allow nodes to activate/deactivate multiple times, effectively being awake for multiple non-contiguous slots each epoch. To ensure that the power consumption model remains accurate in this scenario, we can extend the power model such that a cost is incurred for each activation. Additionally, one can explore adding constraints to manage latency in delivery of information to a sink node. This is naturally expressed as a set of additional constraints on the awake times between parent and children nodes in a collection tree.

To give an example, we consider the problem of maximizing lifetime given a certain level of required coverage. We also extend the model to consider multiple activations of nodes within the epoch, since in this case Theorem 1 no longer applies. For power consumption, we adopt a model in which a node incurs an energy cost proportional to its active interval, with an additional cost for each activation. A conceptual formulation of the problem is the following:

$$\min \quad \max_{n \in \mathcal{N}} \alpha I(n) + \beta W_a(n), \tag{10}$$

subject to

$$< \text{the assigned area is covered} > \tag{11}$$

The cost function to be minimized is the largest of the energy consumed by each node n. Here, α is the energy consumed by a node during an active slot (i.e., for sensing, computation and communication); $I(n)$ is the total number of active slots for n; β is the energy spent for activation; and $W_a(n)$ is the number of times n is activated/deactivated during the epoch. The vector notation in the cost function can be reformulated in scalar form by rephrasing the minimization in terms of an additional decision variable μ constrained as follows:

$$\min \mu, \quad \text{subject to} \quad \forall n \in \mathcal{N},\ \mu \geq \alpha I(n) + \beta W_a(n)$$

Function $I(n)$ is simply given by $I(n) = \sum_{k=0}^{L-1} x_{n,k}$. Function W_a can be computed by accumulating the difference between adjacent scheduling variables:

$$W_a(n) = \sum_{k=0}^{L-1} |x_{n,k} - x_{n,(k+1) \bmod L}|,$$

The above can be linearized using new variables $s_{n,k}$, and the constraints

$$s_{n,k} \geq x_{n,k} - x_{n,((k+1) \bmod L)}$$
$$s_{n,k} \geq x_{n,((k+1) \bmod L)} - x_{n,k}$$

The minimization process ensures that $s_{n,k}$ will equal the absolute value, even if it is not declared as an integer variable. Finally, constraint (11) can be easily expressed by requiring that the area of the covered regions, for each slot, be greater than the given value A_0:

$$\forall k \in [0, L-1], \sum_{\rho \in \mathcal{R}} A(\rho) C_{\rho,k} \geq A_0$$

where the $C_{\rho,k}$ variables are computed using (6) and (7).

A full evaluation of such alternate scenarios is part of our future work.

5 Related Work

Several studies approached the problem of maximizing the WSN lifetime by running sensor nodes on a low duty cycle, while maintaining a high level of performance. Regarding the coverage problem, we can classify existing approaches as centralized techniques, making use of global information about the deployment, and distributed techniques, typically limited by network connectivity but more easily adapting to network dynamics.

Within centralized techniques, Slijepcevic and Potkonjak [2] are among the first to address the problem of maintaining *full coverage* while minimizing power consumption through active/sleep schedule. The problem is solved by partitioning the nodes into disjoint sets, activated one at a time, where each set of nodes completely covers the monitored area. The solution leverages a centralized heuristic of quadratic complexity in the number of nodes, shown to significantly improve over a simulated annealing approach. However, no exact solution is derived in the paper. Cardei et al. propose a method where nodes are divided in sets that are not necessarily disjoint, achieving further improvements [7]. The goal is to maximize the network lifetime by scheduling the activity of nodes while maintaining *full coverage* over a *finite set of points*. The problem is formulated as an integer linear program that, due to its complexity, is only solved through heuristics. Similar formulations are proposed in [8,9]. In the first case, the authors use a two-step procedure to compute the maximal lifetime and to include communication costs. In the second approach, two coordinated optimization problems are solved to determine the subset partitions and their duration. The authors also propose a greedy distributed heuristic, which is however shown to yield solutions that may perform as much as 40% of the optimal.

Although our technique is also based on an mixed integer (boolean) linear program, unlike previous work we do not aim at maintaining full coverage of an area or of a set of points, rather at establishing a periodic schedule that guarantees the *largest total coverage* of an area over a scheduling period given a specified lifetime of the system, in accordance to the problem statement of [1]. By doing so, we are also able to compute the optimal trade-off between coverage and lifetime. An experimental comparison of our approach against previous work is thus difficult, since the optimization objectives differ. In our approach, each node is activated exactly once per epoch, with the latter constrained to be an integer multiple of the awake interval. Cardei et al. show that this choice, compared to allowing nodes to wake-up multiple times in an epoch, is suboptimal in the case of *full coverage* of a set of points [7]. While this applies also to our problem, we have found that for random topologies, the possible increase in the *largest total coverage* is negligible—zero or fractions of a percent. On the other hand, the restriction allows us to greatly improve the performance of the *exact* optimization program, and handle a much larger number of nodes (see Theorem 1).

As for distributed techniques, they typically use information from neighboring nodes to locally compute a schedule. In the simplest form, nodes wake up

regularly and check whether a neighboring node is awake: if so, they go back to sleep to conserve energy. Ye et al. [10] complement this simple scheme with an adaptive sleeping scheme which dynamically determines the duration of the sleep time to optimally maintain a certain degree of coverage. More elaborate schemes take coverage information from neighboring nodes into account, to preserve the *full coverage* of an area. Tian and Georganas propose a technique that proceeds in rounds [3]. At the beginning of a round, each node computes the fraction of its sensed area that is also covered by neighboring nodes, by exchanging position information. When a node determines that it is fully covered by others it goes to sleep for the rest of the round. Hsin and Liu [11] improve on this scheme with a coordinated sleep scheme where the duration of the sleep state is computed as a function of the residual energy of the node and of the neighboring nodes, instead of being fixed by the length of the round. This way, they obtain a more graceful degradation of the overall coverage as nodes fail. Similarly to our distributed solution, Cao et al. [12] use a setup phase where nodes schedule their operation to overlap least in time with nearby nodes covering the same area. This is done by exchanging exact position information and incrementally adjusting the schedule until the procedure converges. More recently, Cărbunar et al. [13] proposed a distributed algorithm for preserving full coverage and computing the coverage boundary. They reduce the problem of finding redundant nodes to that of checking coverage of certain Voronoi diagrams associated to the topology. This result is used to efficiently compute the coverage due to the neighbors, while the diagrams are updated dynamically as nodes fail. Their experiments show a significant improvement over the method in [3].

In contrast, the distributed technique in [1] is extremely simple, and requires neither exact position information nor time synchronization—a significant asset when considering the overall energy budget of a node. Nonetheless, we have shown that the resulting schedules provide a degree of coverage that come very close to the optimum. This is possible since the goal is to maximize the largest total coverage, instead of maintaining full coverage that, again, makes the experimental comparison with previous work difficult. Moreover, we showed that our mathematical formulation is useful to determine the best trade-off between coverage and lifetime, a very valuable input for dimensioning real deployments.

A related problem is how to achieve the best sensor *placement* to cover an area [14]. Solutions rely on integer programming [15], greedy heuristics [16,17,18] and virtual force methods [19]. These complement the temporal spreading investigated here, and can be applied in parallel to achieve further improvements.

6 Conclusions and Future Work

In this paper we presented a way to compute *optimal* schedules for scattering node wake-up times in a WSN. Although here we focused primarily on area coverage, the wake-up scattering problem has practical relevance in many settings, as discussed in the original wake-up scattering paper [1]. Our evaluation through

simulation shows that the decentralized algorithm there presented, albeit very simple, is remarkably efficient, generating schedules whose performance is within 4% to 11% of the optimal ones. The formulation presented in this paper can be used to evaluate the trade-off between coverage and lifetime and, given the small difference between the optimal and distributed schedules, guide engineering decisions in practical deployments. Finally, the modeling framework we presented here is amenable to extension and adaptation towards similar problems. We are currently investigating such extensions, beginning with the inverse problem of determining the lifetime of the system given a desired coverage.

References

1. Giusti, A., Murphy, A.L., Picco, G.P.: Decentralized Scattering of Wake-up Times in Wireless Sensor Networks. In: Langendoen, K.G., Voigt, T. (eds.) EWSN 2007. LNCS, vol. 4373, pp. 245–260. Springer, Heidelberg (2007)
2. Slijepcevic, S., Potkonjak, M.: Power efficient organization of wireless sensor networks. In: Proc. of the IEEE Int. Conf. on Communications (ICC) (June 2001)
3. Tian, D., Georganas, N.D.: A coverage-preserving node scheduling scheme for large wireless sensor networks. In: First ACM Int. Wkshp. on Wireless Sensor networks and Applications (WSNA) (2002)
4. Huang, C., Tseng, Y.: The coverage problem in a wireless sensor network. In: Proc. of the 2^{nd} ACM Int. Conf. on Wireless Sensor Networks and Applications (WSNA 2003) (September 2003)
5. Passerone, R., Palopoli, L.: Aligned schedules are optimal. Technical report, University of Trento (2008)
6. The Gnu Linear Programming Kit, http://www.gnu.org/software/glpk/
7. Cardei, M., Thai, M., Li, Y., Wu, W.: Energy-efficient target coverage in wireless sensor networks. In: Proc. of INFOCOM (2005)
8. Liu, H., et al.: Maximizing lifetime of sensor surveillance systems. IEEE/ACM Trans. on Networking 15(2), 334–345 (2007)
9. Alfieri, A., Bianco, A., Brandimarte, P., Chiasserini, C.F.: Maximizing system lifetime in wireless sensor networks. European Journal of Operational Research 127(1), 390–402 (2007)
10. Ye, F., Zhong, G., Cheng, J., Lu, S., Zhang, L.: PEAS: A robust energy conserving protocol foo long-lived sensor networks. In: 3^{rd} Int. Conf. on Distributed Computing Systems (ICDCS 2003) (May 2003)
11. Hsin, C., Liu, M.: Network coverage using low duty-cycled sensors: Random & coordinated sleep algorithms. In: Proc. of the 3^{th} Int. Symp. on Information Processing in Sensor Networks (IPSN) (2004)
12. Cao, Q., Abdelzaher, T., He, T., Stankovic, J.: Towards optimal sleep scheduling in sensor networks for rare-event detection. In: Proc. of the 4^{th} Int. Symp. on Information Processing in Sensor Networks (IPSN) (April 2005)
13. Carbunar, B., Grama, A., Vitek, J., Carbunar, O.: Redundancy and coverage detection in sensor networks. ACM Trans. on Sensor Networks 2(1), 94–128 (2006)
14. Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M.: Coverage problems in wireless ad-hoc sensor networks. In: Proc. of 20^{th} IEEE INFOCOM, April 2001, pp. 1380–1387 (2001)

15. Chakrabarty, K., Iyengar, S.S., Qi, H., Cho, E.: Grid coverage for surveillance and target location in distributed sensor networks. IEEE Trans. on Computers 51(12), 1448–1453 (2002)
16. Bulusu, N., Estrin, D., Heidemann, J.: Adaptive beacon placement. In: Proc. of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS), April 2001, pp. 489–498 (2001)
17. Howard, A., Mataric, M., Sukhatme, G.: An incremental self-deployment algorithm for mobile sensor networks. Auton. Robots 13(2), 113–126 (2002)
18. Howard, A., Mataric, M., Sukhatme, G.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proc. of 7^{th} Int. Symp. on Distributed Autonomous Robotic Systems (June 2002)
19. Zou, Y., Chakrabarty, K.: Sensor deployment and target localization in distributed sensor networks. IEEE Trans. on Embedded Computing Systems 3(1), 61–91 (2004)

Sundial: Using Sunlight to Reconstruct Global Timestamps

Jayant Gupchup[1], Răzvan Musăloiu-E.[1], Alex Szalay[2], and Andreas Terzis[1]

[1]Computer Science Department; [2]Physics and Astronomy Department
Johns Hopkins University
{gupchup,razvanm,terzis}@jhu.edu, szalay@jhu.edu

Abstract. This paper investigates postmortem timestamp reconstruction in environmental monitoring networks. In the absence of a time-synchronization protocol, these networks use multiple pairs of (local, global) timestamps to retroactively estimate the motes' clock drift and offset and thus reconstruct the measurement time series. We present *Sundial*, a novel offline algorithm for reconstructing global timestamps that is robust to unreliable global clock sources. Sundial reconstructs timestamps by correlating annual solar patterns with measurements provided by the motes' inexpensive light sensors. The surprising ability to accurately estimate the length of day using light intensity measurements enables Sundial to be robust to arbitrary mote clock restarts. Experimental results, based on multiple environmental network deployments spanning a period of over 2.5 years, show that Sundial achieves accuracy as high as 10 parts per million (ppm), using solar radiation readings recorded at 20 minute intervals.

1 Introduction

A number of environmental monitoring applications have demonstrated the ability to capture environmental data at scientifically-relevant spatial and temporal scales [11,12]. These applications do not need online clock synchronization and in the interest of simplicity and efficiency often do not employ one. Indeed, motes do not keep any global time information, but instead, use their local clocks to generate local timestamps for their measurements. Then, a postmortem timestamp reconstruction algorithm retroactively uses (local, global) timestamp pairs, recorded for each mote throughout the deployment, to reconstruct global timestamps for all the recorded local timestamps. This scheme relies on the assumptions that a mote's local clock increases monotonically and the global clock source (e.g., the base-station's clock) is completely reliable. However, we have encountered multiple cases in which these assumptions are violated. Motes often reboot due to electrical shorts caused by harsh environments and their clocks restart. Furthermore, basestations' clocks can be desynchronized due to human and other errors. Finally the basestation might fail while the network continues to collect data.

We present *Sundial*, a robust offline time reconstruction mechanism that operates in the absence of any global clock source and tolerates random mote

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 183–198, 2009.
© Springer-Verlag Berlin Heidelberg 2009

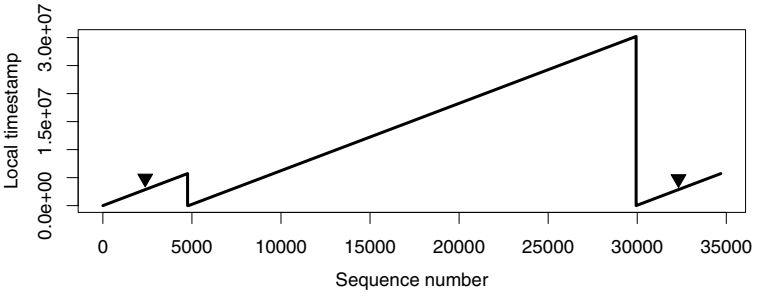


Fig. 1. An illustration of mote reboots, indicated by clock resets. Arrows indicate the segments for which anchor points are collected.

clock restarts. Sundial's main contribution is a novel approach to reconstruct the global timestamps using only the repeated occurrences of day, night and noon. We expect Sundial to work alongside existing postmortem timestamp reconstruction algorithms, in situations where the basestations' clock becomes inaccurate, motes disconnect from the network, or the basestation fails entirely. While these situations are infrequent, we have observed them in practice and therefore warrant a solution. We evaluate Sundial using data from two long-term environmental monitoring deployments. Our results show that Sundial reconstructs timestamps with an accuracy of one minute for deployments that are well over a year.

2 Problem Description

The problem of reconstructing global timestamps from local timestamps applies to a wide range of sensor network applications that correlate data from different motes and external data sources. This problem is related to mote clock synchronization, in which motes' clocks are persistently synchronized to a global clock source. However, In this work, we focus on environmental monitoring applications that do not use online time synchronization, but rather employ postmortem timestamp reconstruction to recover global timestamps.

2.1 Recovering Global Timestamps

As mentioned before, each mote records measurements using its local clock which is not synchronized to a global time source. During the lifetime of a mote, a basestation equipped with a global clock collects multiple pairs of (local, global) timestamps. We refer to these pairs as *anchor points*[1]. Furthermore, we refer to the series of local timestamps as LTS and the series of global timestamps as GTS. The basestation maintains a list of anchor points for each mote and

[1] We ignore the transmission and propagation delays associated with the anchor point sampling process.

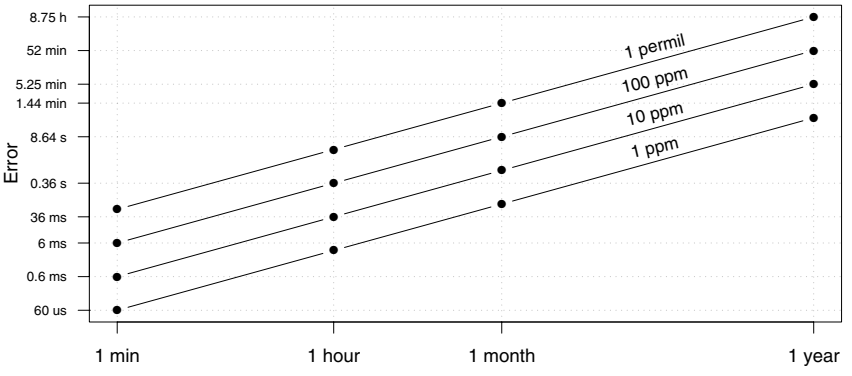


Fig. 2. Time reconstruction error due to α estimation errors as a function of the deployment lifetime

is responsible for reconstructing the global timestamps using the anchor points and the local timestamps.

The mapping between local clock and global clock can be described by the linear relation $GTS = \alpha \cdot LTS + \beta$, where α represents the slope and β represents the intercept (start time). The basestation computes the correct α and β for each mote using the anchor points. Note that these α and β values hold, if and only if the mote does not reboot. In the subsections that follow, we describe the challenges encountered in real deployments where the estimation of α and β becomes non-trivial.

2.2 Problems in Timestamp Reconstruction

The methodology sketched in Section 2.1 reconstructs the timestamps for blocks of measurements where the local clock increase monotonically. We refer to such blocks as *segments*. Under ideal conditions, a single segment includes all the mote's measurements. However, software faults and electrical shorts (caused by moisture in the mote enclosures) are two common causes for unattended mote reboots. The mote's local clock resets after a reboot and when this happens we say that the mote has started a new segment.

When a new segment starts, α and β must be recomputed. This implies that the reconstruction mechanism described above must obtain at least two anchor points for each segment. However, as node reboots can happen at arbitrary times, collecting two anchor points per segment is not always possible. Figure 1 shows an example where no anchor points are taken for the biggest segment, making the reconstruction of timestamps for that segment problematic. In some cases we found that nodes rebooted repeatedly and did not come back up immediately. Having a reboot counter helps recover the segment chronology but does not provide the precise start time of the new segment.

Furthermore, the basestation is responsible for providing the global timestamps used in the anchor points. Our experience shows that assuming the veracity of the basestation clock can be precarious. Inaccurate basestation clocks can corrupt anchor points and lead to bad estimates of α and β introducing errors in timestamp reconstruction. Long deployment exacerbate these problems, as Figure 2 illustrates: an α error of 100 parts per million (ppm) can lead to a reconstruction error of 52 minutes over the course of a year.

2.3 A Test Case

Our *Leakin Park* deployment (referred to as *"L"*) provides an interesting case study of the problems described above. The L deployment comprised six motes deployed in an urban forest to study the spatial and temporal heterogeneity in a typical urban soil ecosystem. The deployment spanned over a year and a half, providing us with half a million measurements from five sensing modalities. We downloaded data from the sensor nodes very infrequently using a laptop PC and collected anchor points only during these downloads. One of the soil scientists in our group discovered that the ambient temperature values did not correlate among the different motes. Furthermore, correlating the ambient temperature with an independent weather station, we found that the reconstruction of timestamps had a major error in it.

Figure 3 shows data from two ambient temperature sensors that were part of the L deployment. Node 72 and 76 show coherence for the period in April, but data from June are completely out-of-sync. We traced the problem back to the laptop acting as the global clock source. We made the mistake of not synchronizing its clock using NTP before going to the field to download the data. As a result the laptop's clock was off by 10 hours, giving rise to large errors in our α and β estimates and thereby introducing large errors in the reconstructed timestamps. To complicate matters further, we discovered that some of the motes had rebooted a few times between two consecutive downloads and we did not have any anchor points for those segments of data.

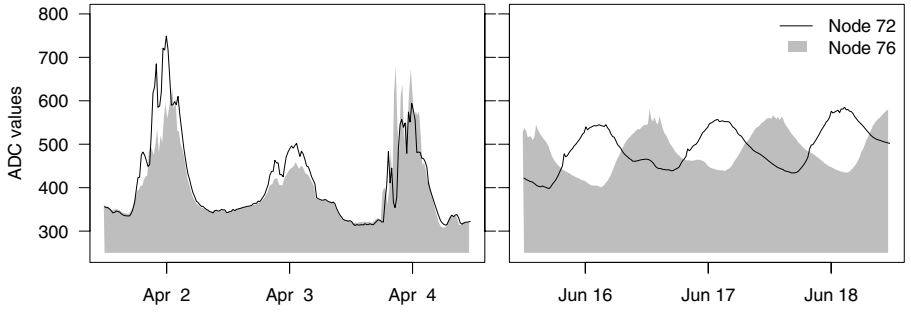


Fig. 3. Ambient temperature data from two motes from the L deployment. The correlation of temperature readings in the left panel indicates consistent timestamps at the segment's start. After two months, the mote's reading become inconsistent due to inaccurate α estimates.

Algorithm 1. Robust Global Timestamp Reconstruction (RGTR)

```
constants
Q                                          ▷ Constant used to identify anchor points for the segment
δ_HIGH, δ_LOW, δ_DEC                       ▷ Constants used in iterative fit

procedure ClockFit(ap)
    (r, i) ← (0, 0)
    q ← HoughQuantize(ap)
    for each γ in Keys(q) do
        s ← Size(q{γ})
        if s > r then
            (r, i) ← (s, γ)
    return ComputeAlphaBeta(q{i})

procedure HoughQuantize(ap)
    q ← {}                                 ▷ Map of empty sets
    for each (lts_i, gts_i) in ap do
        for each (lts_j, gts_j) in ap and (lts_j, gts_j) ≠ (lts_i, gts_i) do
            α ← (gts_j − gts_i)/(lts_j − lts_i)
            if 0.9 ≤ α ≤ 1.1 then          ▷ Check if part of the same segment
                β ← gts_j − α · lts_j
                γ ← ROUND(β/Q)
                Insert(q{γ}, (lts_i, gts_i))
                Insert(q{γ}, (lts_j, gts_j))
    return q

procedure ComputeAlphaBeta(ap)
    δ ← δ_HIGH
    bad ← {}
    while δ > δ_LOW do
        (α, β) ← LLSE(ap)
        for each (lts, gts) ∈ ap and (lts, gts) ∉ bad do
            residual ← (α · lts + β) − gts
            if residual ≥ |δ| then
                Insert(bad, (lts, gts))
        δ ← δ − δ_DEC
    return (α, β)
```

3 Solution

The test case above served as the motivation for a novel methodology that robustly reconstructs global timestamps. The Robust Global Timestamp Reconstruction (RGTR) algorithm, presented in Section 3.1, outlines a procedure to obtain robust estimates of α and β using anchor points that are potentially unreliable. We address situations in which the basestation fails to collect any anchor points for a segment through a novel method that uses solar information alone to generate anchor points. We refer to this mechanism as Sundial.

3.1 Robust Global Timestamp Reconstruction (RGTR)

Having a large number of anchor points ensures immunity from inaccurate ones, provided they are detected. Algorithm 1 describes the Robust Global Timestamp Reconstruction (RGTR) algorithm that achieves this goal. RGTR takes as input a set of anchor points (ap) for a given segment and identifies the anchor points that belong to that segment, while censoring the bad ones. Finally, the algorithm

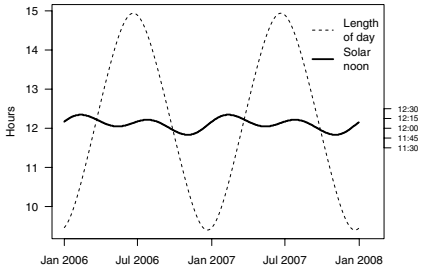


Fig. 4. The solar (model) length of day (LOD) and noon pattern for a period of two years for the latitude of our deployments

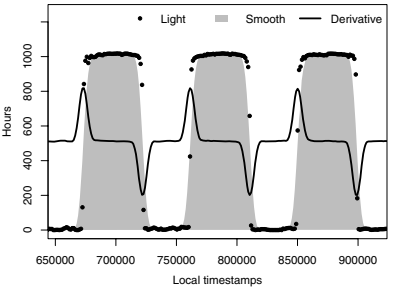


Fig. 5. The light time series (raw and smoothed) and its first derivative. The inflection points represent sunrise and sunset.

returns the (α, β) values for the segment. RGTR assumes the availability of two procedures: INSERT and LLSE. The INSERT(x, y) procedure adds a new element, y, to the set x. The Linear Least Square Estimation [4], LLSE procedure takes as input a set of anchor points belonging to the same segment and outputs the parameters (α, β) that minimize the sum of square errors.

RGTR begins by identifying the anchor points for the segment. The procedure HOUGHQUANTIZE implements a well known feature extraction method, known as the Hough Transform [5]. The central idea of this method is that anchor points that belong to the same segment should fall on a straight line having a slope of ~ 1.0. Also, if we consider pairs of anchors (two at a time) and quantize the intercepts, anchors belonging to the same segment should all collapse to the same quantized value (bin). HOUGHQUANTIZE returns a map, q, which stores the anchor points that collapse to the same quantized value. The key (stored in i) that contains the maximum number of elements contains the anchor points for the segment.

Next, we invoke the procedure COMPUTEALPHABETA to compute robust estimates of α and β for a given segment. We begin by creating an empty set, *bad*. The set *bad* maintains a list of all anchor points that are detected as being outliers and do not participate in the parameter estimation. This procedure is iterative and begins by estimating the fit (α, β) using all the anchor points. Next, we look at the residual of all anchor points with the fit. Anchor points whose residuals exceed the current threshold, δ, are added to the *bad* set and are excluded in the next iteration fit. Initially, δ is set conservatively to δ_{HIGH}. At the end of every iteration, the δ threshold is lowered and the process repeats until no new entries are added to the *bad* set, or δ reaches δ_{LOW}.

3.2 Sundial

The parameters of the solar cycle (sunrise, sunset, noon) follow a well defined pattern for locations on Earth with a given latitude. This pattern is evident in

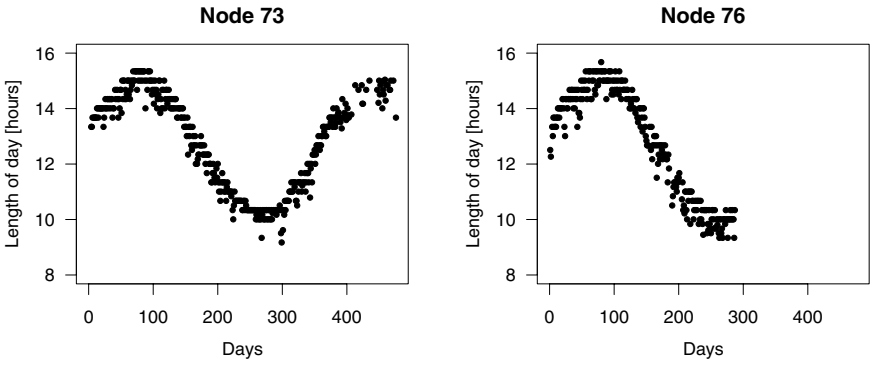


Fig. 6. The length of day pattern for two long segments belonging to different nodes. Day 0 represents the start-time for each of the segments

Figure 4 that presents the length of day (LOD) and solar noon for the period between January 2006 and June 2008 for the latitude of the L deployment. Note that the LOD signal is periodic and sinusoidal. Furthermore, the frequency of the solar noon signal is twice the frequency of the LOD signal. We refer the reader to [6] for more details on how the length of day can be computed for a given location and day of the year.

The paragraphs that follow explain how information extracted from our light sensors can be correlated with known solar information to reconstruct the measurement timestamps.

Extracting Light Patterns: We begin by looking at the time series L_i of light sensor readings for node i. L_i is defined for a single segment in terms of the local clock. First, we create a smooth version of this series, to remove noise and sharp transients. Then, we compute the first derivative for the smoothed L_i series, generating the D_i time-series. Figure 5 provides an illustration of a typical D_i series overlaid on the light sensor series (L_i). One can notice the pattern of inflection points representing sunrise and sunset. The regions where the derivative is high represent mornings, while the regions where the derivative is low represent evenings. For this method, we select sunrise to be the point at which the derivative is maximum and sunset the point at which the derivative is minimum. Then, LOD is given as the difference between sunrise and sunset, while noon is set to the midpoint between sunrise and sunset.

The method described above accurately detects noon time. However, the method introduces a constant offset in LOD detection and it underestimates LOD due to a late sunrise detection and an early sunset detection. The noon time is unaffected due to these equal but opposite biases. In practice, we found that a simple thresholding scheme works best for finding the sunrise and sunset times. The light sensors' sensitivity to changes simplifies the process of selecting the appropriate threshold. In the end, we used a hybrid approach whereby we obtain noon times from the method that uses derivatives and LOD times from the thresholding method. The net result of this procedure is a set of noon

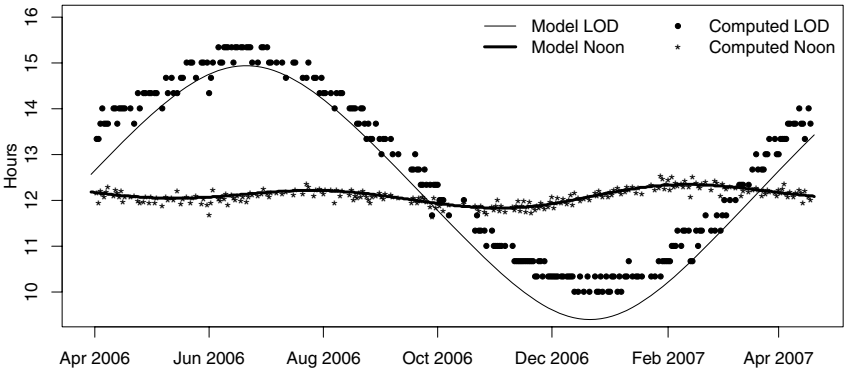


Fig. 7. An illustration of the computed LOD and noon values for the lag with maximum correlation with the solar model

times and LOD for each day from the segment's start in terms of the local clock. Figure 6 shows the LOD values obtained for two different node segments after extracting the light patterns.

Solar Reconstruction of Clocks: The solar model provides the LOD and noon values in terms of the global clock (LOD_{GT}), while the procedure described in the previous paragraph extracts the LOD and noon values from light sensor measurements in terms of the motes' local clocks (LOD_{LT}). In order to find the best possible day alignment, we look at the correlation between the two LOD signals (LOD_{GT}, LOD_{LT}) as a function of the lag (shift in days). The lag that gives us the maximum correlation (ρ_{max}) is an estimate of the day alignment. Mathematically, the day alignment estimate (lag) is obtained as

$$\arg\max_{lag} \mathrm{Cor}(LOD_{GT}, LOD_{LT}, lag)$$

where $\mathrm{Cor}(X, Y, s)$ is the correlation between time series X and Y shifted by s time units. Figure 7 presents an example of the match between model and computed LOD and noon times achieved by the lag with the highest correlation. The computed LOD time series tracks the one given by the solar model. One also observes a constant shift between the two LOD patterns, which can be attributed to the horizon effect. For some days, canopy cover and weather patterns cause the extracted LOD to be underestimated. However, as the day alignment is obtained by performing a cross-correlation with the model LOD pattern, the result is robust to constant shifts. Furthermore, Figure 7 shows that the equal and opposite effect of sunrise and sunset detection ensures that the noon estimation in unaffected in the average case.

After obtaining the day alignment, we use the noon information to generate anchor points. Specifically, for each day of the segment we have available to us the noon time in local clock (from the light sensors) and noon time in global clock (using the model). RGTR can then be used to obtain robust values of

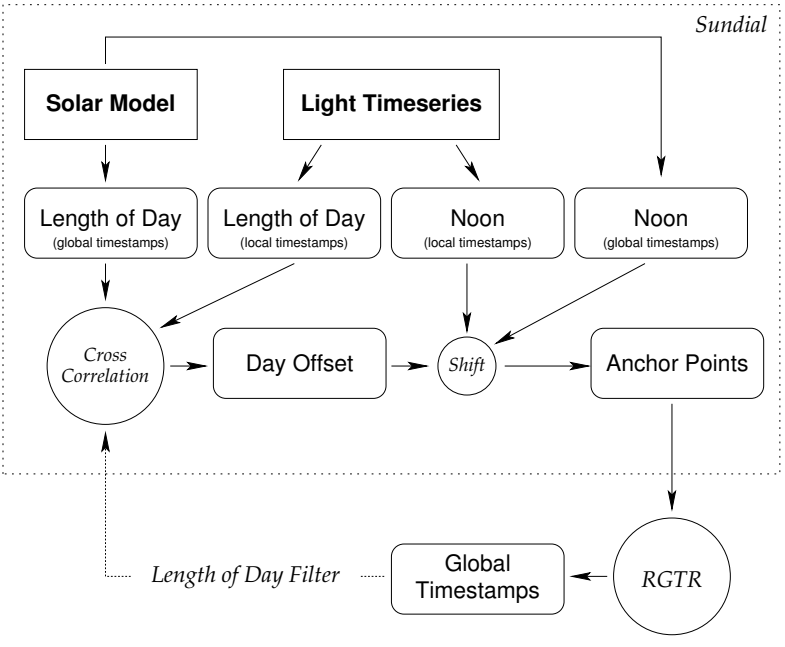


Fig. 8. The steps involved in reconstructing global timestamps using Sundial

α and β. This fit is used to reconstruct the global timestamps. As Figure 4 suggests, the noon times change slowly over consecutive days as they oscillate around 12:00. Thus, even if the day estimate is inaccurate, due to the small difference in noon times, the α estimate remains largely unaffected. This implies that even if the day alignment is not optimal, the time reconstruction within the day will be accurate, provided that the noon times are accurately aligned. The result of an inaccurate lag estimate is that β is off by a value equal to the difference between the actual day and our estimate. In other words, β is off by an integral and constant number of days (without any skew) over the course of the whole deployment period.

We find that this methodology is well suited in finding the correct α. To improve the β estimate, we perform an iterative procedure which works as follows. For each iteration, we obtain the best estimate fit (α, β). We convert the motes' local timestamps into global timestamps using this fit. We then look at the difference between the actual LOD (given by the model) and the current estimate for that day. If the difference between the expected LOD and the estimate LOD exceeds a threshold, we label that day as an outlier. We remove these outliers and perform the LOD cross-correlation to obtain the day shift (lag) again. If the new lag differs from the lag in the previous iteration, a new fit is obtained by shifting the noon times by an amount proportional to the new lag. We iterate until the lag does not change from the previous iteration. Figure 8 shows a schematic of the steps involved in reconstructing global timestamps for a segment.

4 Evaluation

We evaluate the proposed methodology using data from two deployments. Deployment J was done at the Jug Bay wetlands sanctuary along the Patuxent river in Anne Arundel County, Maryland. The data it collected is used to study the nesting conditions of the Eastern Box turtle (*Terrapene carolina*) [10]. Each of the motes was deployed next to a turtle nest, whereas some of them have a clear view of the sky while others are under multiple layers of tree canopy. Deployment L, from Leakin Park, is described in Section 2.3.

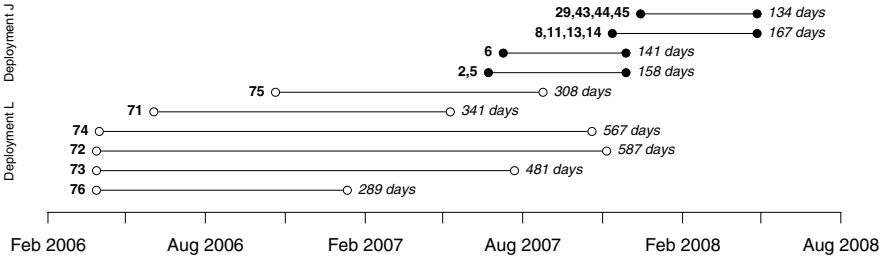


Fig. 9. Node identifiers, segments and length of each segment (in days) for the two deployments used in the evaluation

Figure 9 summarizes the node identifiers, segments, and segment lengths in days for each of the two deployments. Recall that a segment is defined as a block of data for which the mote's clock increases monotonically. Data obtained from the L dataset contained some segments lasting well over 500 days. The L deployment uses MicaZ motes [3], while the J deployment uses TelosB motes [8]. Motes 2, 5, and 6 from Deployment J collected samples every 10 minutes. All other motes for both deployments had a sampling interval of 20 minutes. In addition to its on-board light, temperature, and humidity sensors, each mote was connected to two soil moisture and two soil temperature sensors.

In order to evaluate Sundial's accuracy, we must compare the reconstructed global timestamps it produces, with timestamps that are known to be accurate and precise. Thus we begin our evaluation by establishing the ground truth.

4.1 Ground Truth

For each of the segments shown in Figure 9, a set of good anchor points (sampled using the basestation) were used to obtain a fit that maps the local timestamps to the global timestamps. We refer to this fit as the *Ground truth fit*. This fit was validated in two ways. First, we correlated the ambient temperature readings among different sensors. We also correlated the motes' measurements with the air temperature measurements recorded by nearby weather stations. The weather station for the L deployment was located approximately 17 km

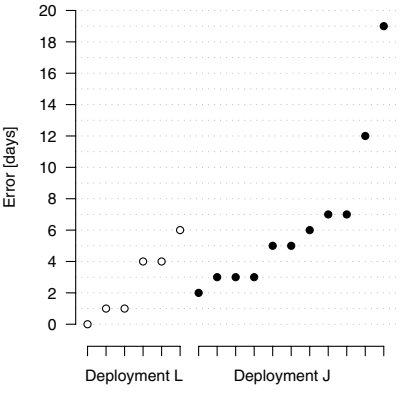


Fig. 10. Error in days for different motes from the L and J deployments

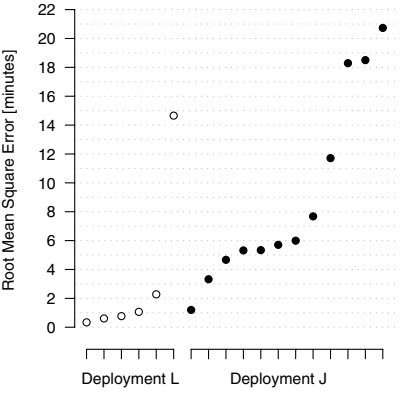


Fig. 11. Root mean square error in minutes ($RMSE_{min}$)

away from the deployment site [1], while the one for the J deployment was located less than one km away [7]. Considering the proximity of the two weather stations we expect that their readings are strongly correlated to the motes' measurements.

Note that even if the absolute temperature measurements differ, the diurnal temperature patterns should exhibit the same behavior thus leading to high correlation values. Visual inspection of the temperature data confirmed this intuition. Finally, we note that due to the large length of the segments we consider, any inconsistencies in the ground truth fit would become apparent for reasons similar to the ones provided in Section 2.2.

4.2 Reconstructing Global Timestamps Using Sundial

We evaluate Sundial using data from the segments shown in Figure 9. Specifically, we evaluate the accuracy of the timestamps reconstructed by Sundial as though the start time of these segment is unknown (similar to the case of a mote reboot) and no anchor points are available. Since we make no assumptions of the segment start-time, a very large model (solar) signal needs to be considered to find the correct shift (lag) for the day alignment.

Evaluation Metrics: We divide the timestamp reconstruction error to: (a) error in days; and (b) error in minutes within the day. The error in minutes is computed as the root mean square error ($RMSE_{min}$) over all the measurements. We divide the reconstruction error into these two components, because this decoupling naturally reflects the accuracy of estimating the α and β parameters. Specifically, if the α estimate were inaccurate, then, as Figure 2 suggests, the reconstruction error would grow as a function of time. In turn, this would result in a large root mean squared error in minutes within the day over all the

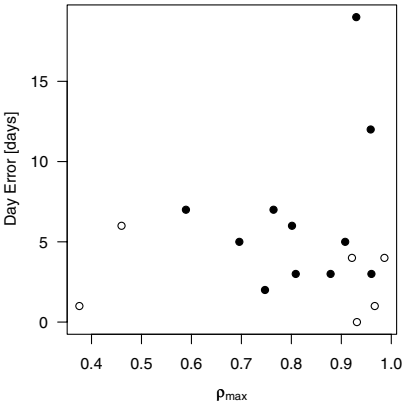


Fig. 12. Relation between ρ_{max} and error in days

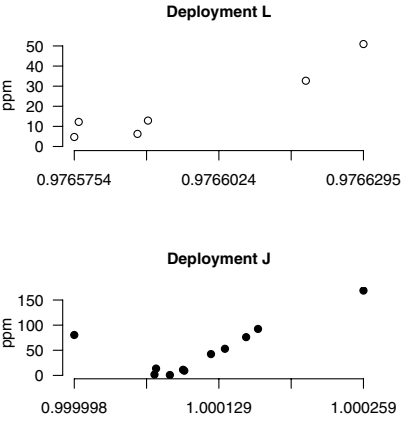


Fig. 13. α estimates from Sundial and estimation errors in ppm

measurements. On the other hand, a low $RMSE_{min}$ corresponds to an accurate estimate for α. Likewise, inaccuracies in the estimation of β would result in large error in days.

Results: Figures 10 and 11 summarize Sundial's accuracy results. Overall, we find that longer segments show a lower day error. Segments belonging to the L deployment span well over a year and the minimum day error is 0 while the maximum day error is 6. In contrast, most of the segments for deployment J are less than 6 months long and the error in days for all but two of those segments is less than one week. Figure 12 presents the relationship between the maximum correlation (ρ_{max}) and the day error. As ρ_{max} measures how well we are able to match the LOD pattern for a node with the solar LOD pattern, it is not surprising that high correlation is generally associated with low reconstruction error. The $RMSE_{min}$ obtained for each of the segments in deployment L is very low (see Figure 11) . Remarkably, we are able to achieve an accuracy ($RMSE_{min}$) of under a minute for the majority of the nodes of the L deployment even though we are limited by our sampling frequency of 20 minutes. Moreover, $RMSE_{min}$ error is always within one sample period for all but one segment.

Interestingly, we found that the α values for the two deployments were significantly different. This disparity can be attributed to differences in node types and thus clock logic. Nonetheless, Sundial accurately determined α in both cases. Figure 13 presents the α values for the two deployments. We also show the error between the α obtained using Sundial and the α value obtained by fitting the good anchor points sampled by the gateway (i.e., ground truth fit). The ppm error for both the deployments is remarkably low and close to the operating error of the quartz crystal.

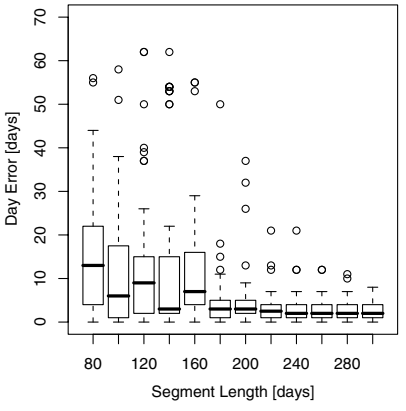


Fig. 14. Error in days as a function of segment size

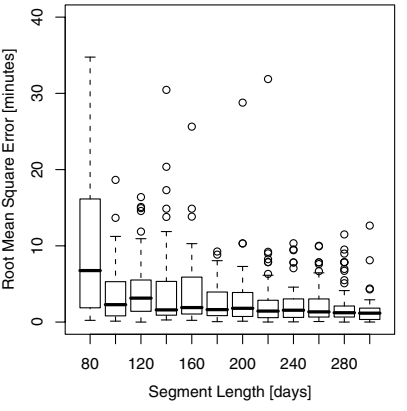


Fig. 15. Error in minutes ($RMSE_{min}$) as a function of segment size

4.3 Impact of Segment Length

Sundial relies on matching solar patterns to the ones observed by the light sensors. The natural question to ask is: what effect does the length of segment have on the reconstruction error. We address this question by experimenting with the length of segments and observing the reconstruction error in days and $RMSE_{min}$. We selected data from three long segments from deployment L. To eliminate bias, the start of each shortened segment was chosen from a uniform random distribution. Figure 15 shows that the $RMSE_{min}$ tends to be remarkably stable even for short segments. One concludes that even for short segment lengths, Sundial estimates the clock drift (α) accurately. Figure 14 shows the effect of segment size on day error. In general, the day error decreases as the segment size increases. Moreover, for segments less than 150 days long, the error tends to vary considerably.

4.4 Day Correction

The results so far show that 88% (15 out of 17) of the motes have a day offset of less than a week. Next, we demonstrate how global events can be used to correct for the day offset. We looked at soil moisture data from eight motes of the J deployment after obtaining the best possible timestamp reconstruction. Specifically, we correlated the motes' soil moisture data with rainfall data to correct for the day offset. We used rainfall data from a period of 133 days, starting from December 4, 2007, during which 21 major rain events occurred. To calculate the correlation, we created weighted daily vectors for soil moisture measurements (SM) whose value was greater than a certain threshold and similarly rainfall vectors having a daily precipitation (PPT) value of greater than 4.0 cm. Next, we extracted the lag at which the cosine angle between the two vectors (cosine similarity, θ_{SM-PPT}) is maximum. This method is inspired by the well-known

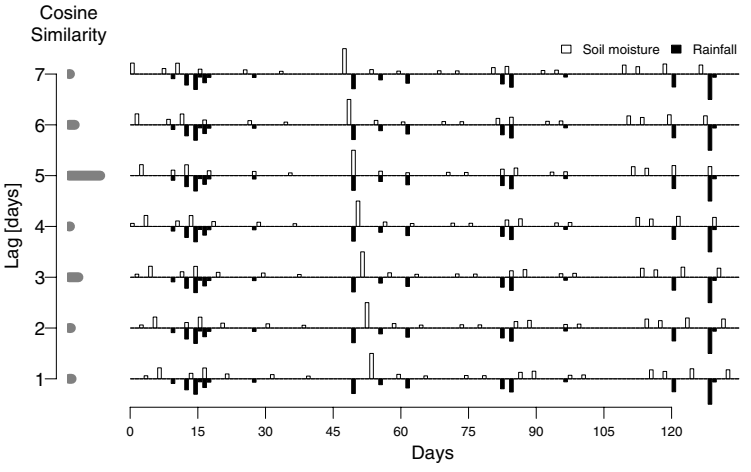


Fig. 16. An illustration of the cosine similarity (θ_{SM-PPT}) values for seven different day lags between moisture and rainfall vectors. θ_{SM-PPT} peaks at the correct lag of five days, providing the correct day adjustment.

document clustering model used in the information retrieval community [9]. Note that we computed θ_{SM-PPT} for a two-week window ($\pm$ seven days) of lags and found that seven out of the eight motes could be aligned perfectly. Figure 16 illustrates the soil moisture vectors, rainfall vectors and the associated θ_{SM-PPT} for seven lags for one of the segments. Note that θ_{SM-PPT} peaks at the correct lag of five, leading to the precise day correction. While we use soil moisture to illustrate how global events can be used to achieve macro-level clock adjustments, other modalities can also be used based on the application's parameters.

5 Related Work

This study proposes a solution to the problem of postmortem timestamp reconstruction for sensor measurements. To our knowledge, there is little previous work that addresses this problem for deployments that span a year or longer. Deployment length can be an issue because the reconstruction error monotonically increases as a function of time (cf. Sec. 2.2). The timestamp reconstruction problem was first introduced by Werner-Allen et al. who provided a detailed account of the challenges they faced in synchronizing mote clocks during a 19-day deployment at an active volcano [13]. Specifically, while the system employed the FTSP protocol to synchronize the network's motes, unexpected faults forced the authors to rely on an offline *time rectification* algorithm to reconstruct global timestamps.

While experiences such as the one reported in [13] provide motivation for an independent time reconstruction mechanism such as the one proposed in this paper, the problem addressed by Werner-Allen et al. is different from the one we aim to solve. Specifically, the volcano deployment had access to precise

global timestamps (through a GPS receiver deployed at the site) and used linear regression to translate local timestamps to global time, once timestamp outliers were removed. While RGTR can also be used for outlier detection and timestamp reconstruction, Sundial aims to recover timestamps in situations where a reliable global clock source is not available.

Finally, Chang et. al. [2] describe their experiences with motes rebooting and resetting of logical clocks, but do not furnish any details of how they reconstructed the global timestamps when this happens.

6 Conclusion

In this paper we present Sundial, a method that uses light sensors to reconstruct global timestamps. Specifically, Sundial uses light intensity measurements, collected by the motes' on-board sensors, to reconstruct the length of day (LOD) and noon time throughout the deployment period. It then calculates the slope and the offset by maximizing the correlation between the measurement-derived LOD series and the one provided by astronomy. Sundial operates in the absence of global clocks and allows for random node reboots. These features make Sundial very attractive for environmental monitoring networks deployed in harsh environments, where they operate disconnected over long periods of time. Furthermore, Sundial can be used as an independent verification technique along with any other time reconstruction algorithm.

Using data collected by two network deployments spanning a total of 2.5 years we show that Sundial can achieve accuracy in the order of a few minutes. Furthermore, we show that one can use other global events such as rain events to correct any day offsets that might exist. As expected, Sundial's accuracy is closely related to the segment size. In this study, we perform only a preliminary investigation on how the length of the segment affects accuracy. An interesting research direction we would like to pursue is to study the applicability of Sundial to different deployments. Specifically, we are interested in understanding how sampling frequency, segment length, latitude and season (time of year) collectively affect reconstruction accuracy.

Sundial exploits the correlation between the well-understood solar model and the measurements obtained from inexpensive light sensors. In principle, any modality having a well-understood model can be used as a replacement for Sundial. In the absence of a model, one can exploit correlation from a trusted data source to achieve reconstruction, e.g., correlating the ambient temperature measurement between the motes with data obtained from a nearby weather station. However, we note that many modalities (such as ambient temperature) can be highly susceptible to micro-climate effects and exhibit a high degree a spatial and temporal variation. Thus, the micro-climate invariant solar model makes light a robust modality to reconstruct timestamps in the absence of any sampled anchor points.

Finally, we would like to emphasize the observation that most environmental modalities are affected by the diurnal and annual solar cycles and not by the human-created universal time. In this regard, the time base that Sundial establishes offers a more natural reference basis for environmental measurements.

Acknowledgments

We would like to thank Yulia Savva (JHU, Department of Earth and Planetary Science) for helping us identify the timestamp reconstruction problem. This research was supported in part by NSF grants CNS-0546648, CSR-0720730, and DBI-0754782. Any opinions, finding, conclusions or recommendations expressed in this publication are those of the authors and do not represent the policy or position of the NSF.

References

1. Baltimore-Washington International airport, weather station, http://weather.marylandweather.com/cgi-bin/findweather/getForecast?query=BWI
2. Chang, M., Cornou, C., Madsen, K., Bonett, P.: Lessons from the Hogthrob Deployments. In: Proceedings of the Second International Workshop on Wireless Sensor Network Deployments (WiDeploy 2008) (June 2008)
3. Crossbow Corporation. MICAz Specifications, http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf
4. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, Chichester (2001)
5. Duda, R.O., Hart, P.E.: Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM 15(1), 11–15 (1972)
6. Forsythea, W.C., Rykiel Jr., E.J., Stahla, R.S., Wua, H., Schoolfield, R.M.: A model comparison for daylength as a function of latitude and day of year. ScienceDirect 80(1) (January 1994)
7. National Estuarine Research Reserve. Jug Bay weather station (cbmjbwq), http://cdmo.baruch.sc.edu/QueryPages/anychart.cfm
8. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling Ultra-Low Power Wireless Research. In: Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS) (April 2005)
9. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 18(11), 613–620 (1975)
10. Szlavecz, K., Terzis, A., Musaloiu-E., R., Liang, C.-J., Cogan, J., Szalay, A., Gupchup, J., Klofas, J., Xia, L., Swarth, C., Matthews, S.: Turtle Nest Monitoring with Wireless Sensor Networks. In: Proceedings of the American Geophysical Union, Fall Meeting (2007)
11. Terzis, A., Musaloiu-E., R., Cogan, J., Szlavecz, K., Szalay, A., Gray, J., Ozer, S., Liang, M., Gupchup, J., Burns, R.: Wireless Sensor Networks for Soil Science. International Journal on Sensor Networks
12. Tolle, G., Polastre, J., Szewczyk, R., Turner, N., Tu, K., Buonadonna, P., Burgess, S., Gay, D., Hong, W., Dawson, T., Culler, D.: A Macroscope in the Redwoods. In: Proceedings of the 3^{rd} ACM SenSys Conference (November 2005)
13. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and Yield in a Volcano Monitoring Sensor Network. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (November 2006)

An Analytical Study of Reliable and Energy-Efficient Data Collection in Sparse Sensor Networks with Mobile Relays

Giuseppe Anastasi[1], Marco Conti[2], and Mario Di Francesco[1]

[1] Dept. of Information Engineering, University of Pisa, Italy
{giuseppe.anastasi,mario.difrancesco}@iet.unipi.it
[2] CNR-IIT, National Research Council, Italy
marco.conti@iit.cnr.it

Abstract. Sparse wireless sensor networks (WSNs) are emerging as an effective solution for a wide range of applications, especially for environmental monitoring. In this context, special mobile elements – i.e. mobile relays (MRs) – can be used to get data sampled by sensor nodes. In this paper we present an analytical evaluation of the data collection performance in sparse WSNs with MRs. Our main contribution is the definition of a flexible model which can derive the total energy consumption for each message correctly transferred by sensors to the MR. The results show that a low duty cycle is convenient and allows a significant amount of correctly received messages, especially when the MR moves with a low speed. When the MR moves fast, depending on its mobility pattern, a low duty cycle may not always be the most energy efficient option.

1 Introduction

Wireless sensor networks (WSNs) have become an enabling technology for a wide range of applications. The traditional WSN architecture consists of a large number of sensor nodes which are densely deployed over an area of interest. Sensor nodes sample data from their surrounding environment, process them locally and send the results to a data collection point, usually a sink node or an Access Point (AP). The communication between the sensors and the data collection point is multi-hop, which is possible due to the network density. More recently, a different WSN architecture has been introduced for application scenarios where fine-grained sensing is not required. In this case, nodes are sparsely deployed over the sensing field. As the number of nodes is moderate or low, in contrast with traditional solutions, the costs are reduced. However, since the network is sparse, neighboring nodes are far away from each other, so that they cannot communicate together directly nor through multi-hop paths and a different data gathering scheme is required.

In sparse sensor networks, data collection can be accomplished by means of *mobile relays* (MRs). MRs are special mobile nodes which are responsible for data gathering. They are assumed to be powerful in terms of data storage and

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 199–215, 2009.
© Springer-Verlag Berlin Heidelberg 2009

processing capabilities, and not energy constrained, in the sense that their energy source can be replaced or recharged easily. MRs carry data from sensors to the sink node or an infra-structured AP [1]. Depending on the application scenario, MRs may be either part of the external environment [2,3] (e.g., buses, cabs, or walking people), or part of the network infrastructure [4,5] (e.g., mobile robots).

The communication between an MR and sensor nodes takes place in two different phases. First, sensor nodes have to discover the presence of the MR in their communication range. Then, they can transfer collected data to the MR while satisfying certain reliability constraints, if required. Different from the MRs, sensor nodes have a limited energy budget, so that both discovery and data transfer should be energy efficient in order to prolong the network lifetime [6]. As the radio component is usually the major source of energy consumption, the overall radio activity should be minimized. To this end, a duty cycle approach can be used, so that sensors alternate sleep and active periods. However, the effects of the duty cycle have to be properly investigated: if sensor nodes are sleeping when the MR comes, they cannot detect it neither transmit data, so that they are only wasting their energy.

In this paper we consider the joint impact of discovery and data transfer for reliable and energy efficient data collection in sparse WSNs with MRs. The major contribution of this paper is a detailed and realistic model for deriving the performance of the overall data collection process. The proposed methodology is general, so that it can be adapted to different discovery and data transfer protocols, and does not depend on the mobility pattern of the MR. To the purposes of our analysis, we consider a discovery scheme based on periodic wakeups and an ARQ-based data transfer protocol. Finally, we derive the performance of data collection in terms of both throughput (i.e. average number of messages correctly transferred to the MR) and energy efficiency (i.e. total energy spent per successfully transferred message) at each contact.

The results obtained show that, in general, a low duty cycle provides a better energy efficiency, especially if the contact time is large enough to allow the reliable transfer of a significant amount of data. However, when the contact time is limited, a very low duty cycle is not convenient as the energy saved during discovery is overcome by the decrease in the number of messages successfully transferred.

The rest of the paper is organized as follows. Section 2 presents an overview of the relevant literature in the field. Section 3 introduces the system model and the related assumptions. Section 4 and 5 present the analytical model for the discovery and the data transfer phases, respectively. Section 6 presents and discusses the obtained results. Finally, section 7 concludes the paper, giving insights for future work.

2 Related Work

Many different papers have addressed the issues of data collection using MRs. In the context of opportunistic networks, the well known message ferrying approach has been proposed in [7]. Specifically, power management has been addressed

by [8], where a general framework for energy conservation is introduced. The proposed solution, which can also exploit knowledge about the mobility pattern of the MR, is evaluated in terms of energy efficiency and delivery performance. However, as the proposed solution is devised for opportunistic networks, it is not applicable in the scenario considered in this paper.

Indeed, many solutions have also been conceived specifically for WSNs. While many papers focus on the mobility of the MR [9,10], some works actually address the problem of energy efficient data collection from the sensor node perspective. For example, [5] considers a periodic wakeup scheme for discovery and a stop-and-wait protocol for data transfer. A stop-and-wait protocol for data transfer is also used in [11], where the MR is assumed to be controllable. A different solution is investigated in [2], under the assumption that the MR has a completely predictable mobility. The above mentioned solutions, however, have only been analyzed with simulations, while in this paper we address the problem analytically. In addition, the solution proposed here is flexible enough to support different protocols and mobility patterns of the MR.

In the specific context of MRs, [3] considers MRs which are not controllable but move randomly, and models the success rate of messages arriving at the access point. However, [3] focuses on buffer requirements at sensors rather than on their energy consumption. Under the same scenario, [1] introduces a more detailed formulation, which considers both the discovery and the data transfer phases of data collection. Furthermore, it evaluates different mobility patterns of the MR and supports sensor nodes operating with a duty cycle during discovery. Although discussing the probability of data reception at the access point, both [3] and [1] assume an ideal channel and no specific data transfer protocol, so that their findings are mostly affected by buffering constraints. Instead, we explicitly consider data transfer – in addition to discovery – for reliable data collection. In addition, we take the message loss into account by using a model derived from real measurements.

The problem of reliable and energy efficient data collection has also been addressed in [12], where an adaptive and window-based ARQ transmission scheme is evaluated under a realistic message loss model derived from real measurements [13]. In detail, [12] analytically shows that the proposed scheme achieves not only a better throughput, but also a higher energy efficiency than a simple stop-and-wait protocol. However, the proposed approach is evaluated only in the application scenario where the sensor has only a limited amount of data to send. In addition, [12] does not consider the effect of discovery on the subsequent data transfer phase, as it focuses only on data transfer. On the contrary, the model presented in this paper jointly considers discovery and data transfer for deriving the overall energy efficiency.

3 System Model

In this section, we will first introduce the reference scenario considered in the analysis. Then, we will describe the discovery and the data transfer protocols used by the MR and the static sensors for data collection.

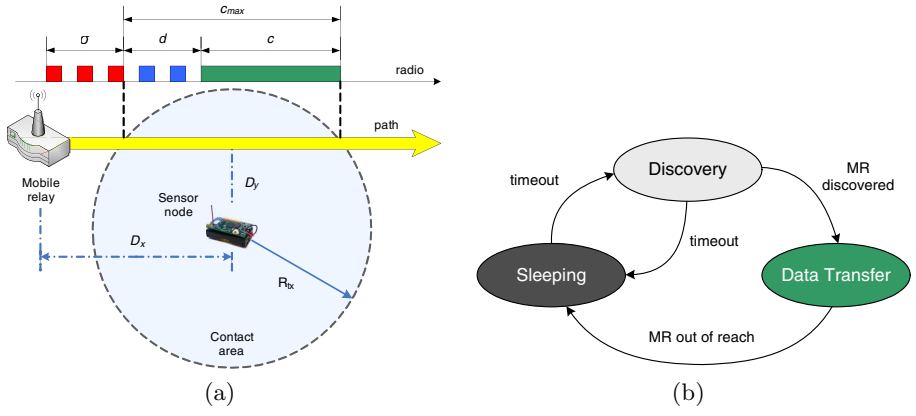


Fig. 1. Reference scenario (a) and state diagram for the static node (b)

The reference network scenario is illustrated in Fig. 1(a). In the following, we will consider a single MR and assume that the network is sparse enough so that at any time at most a single (static) node can reach the MR. In addition, we will assume that the MR is part of the environment (e.g. a bus or a car), so that its mobility pattern cannot be controlled. Finally, we will assume that the MR moves along a linear path at a fixed vertical distance (D_y) from the static node, at a constant speed v. This assumption is reasonable for a sample scenario where the MR moves along a street or a road. Data collection takes place only during a *contact*, i.e. when the static node and the MR can reach each other. Furthermore, the area within the radio transmission range R_{tx} of the static node is called *contact area*[1], and the overall time spent by the MR inside the contact area is called *contact time*, and is referred to as c_{max}. During a contact, messages exchanged between the MR and the static node experience a certain message loss. We denote by $p(t)$ the probability that a message transmitted at time t is lost, and assume as $t = 0$ the instant at which the MR enters the contact area. Any message transmitted when the static node and the MR are not in contact is assumed to get lost, so that $p(t)$ is defined only within the contact area.

The overall data collection process can be split into three main phases. Fig. 1(b) shows the state diagram of the static sensor node [8]. As MR arrivals are generally unpredictable, the static node initially performs a discovery phase for the timely detection of the MR. Indeed, the successful MR detection by the static sensor is not immediate, but requires a certain amount of time, called *discovery time*, and denoted by d in Fig. 1(a). Upon detecting the MR, the static node switches from the discovery state to the data transfer state, and starts transmitting data to the MR. As a result of the discovery process, the static node cannot exploit the whole available contact time for data transfer.

[1] Depicted with a circular shape in Fig. 1(a) only for convenience.

The portion of the contact time which can be actually used for subsequent data transfer is called *residual contact time* and is referred to as c. After the end of the data transfer phase, the static node may switch to the discovery state again in order to detect the next MR passage. However, if the MR has a (even partially) predictable mobility, the static node can exploit this knowledge to further reduce its energy consumption [8]. In this case, the static node can go to a sleep state until the next expected arrival of the MR. In any case, the static sensor may be awake also when the MR is out of reach. The amount of time spent by the static node in the discovery state while the MR has not yet entered the contact area is called *waiting time*, and is indicated with σ in Fig. 1(a).

We now briefly describe the discovery and data transfer protocols used by the static sensor and the MR in the corresponding phases. In principle, different discovery and data transfer protocols could be used for data collection in the above scenario. In our analysis, however, we will consider a discovery protocol based on a periodic beacon transmission by the MR, and an ARQ-based protocol for reliable data transfer, as they are among the most frequent schemes in the field [5,3,11]. To advertise its presence in the surrounding area, the MR periodically sends special messages called *beacons*. The duration of a beacon message is equal to T_{BD}, and subsequent beacons are spaced by a *beacon period*, indicated with T_B. In order to save energy during the discovery phase, the static node operates with a duty cycle δ, defined by the active time T_{ON} and the sleep time T_{OFF}, i.e. $\delta = T_{ON}/(T_{ON} + T_{OFF})$. The static node follows a periodic wakeup scheme, with its activity time set to $T_{ON} = T_B + T_{BD}$, i.e. a value which allows the node to receive a complete beacon during its active time, provided that it wakes up when the MR is in the contact area. On the other hand, the sleep time T_{OFF} is set to a value which enforces the desired duty cycle δ.

As soon as it receives a beacon from the MR, the static node enters the data transfer state. While in this state, the static node remains always active to exploit the residual contact time as much as possible. On the other hand, the MR enters the data transfer phase as soon as it receives the first message sent by the static node, and stops beacon transmissions. The communication scheme adopted during the data transfer phase is based on Automatic Repeat reQuest (ARQ). The static node splits buffered data into messages, which are transmitted in groups (windows). The number of messages contained in a window, i.e. the *window size*, is assumed to be fixed and known both at the sender and at the receiver. The static node sends the messages in a window back to back, then waits for an acknowledgement sent back by the MR. Note that, in this context, the acknowledgement message is used not only for implementing a retransmission strategy, but also as an indication of the MR presence in the contact area. In the following, we assume that the static sensor has always data to send, so that the data transfer phase ends when the MR is not reachable any more (i.e. at the end of the residual contact time). However, the static node generally cannot know when the MR will leave the contact area, for instance because it cannot derive the residual contact time a priori. In practice, the static node assumes that the MR has exited the contact area when it misses N_{ack} consecutive acknowledgments.

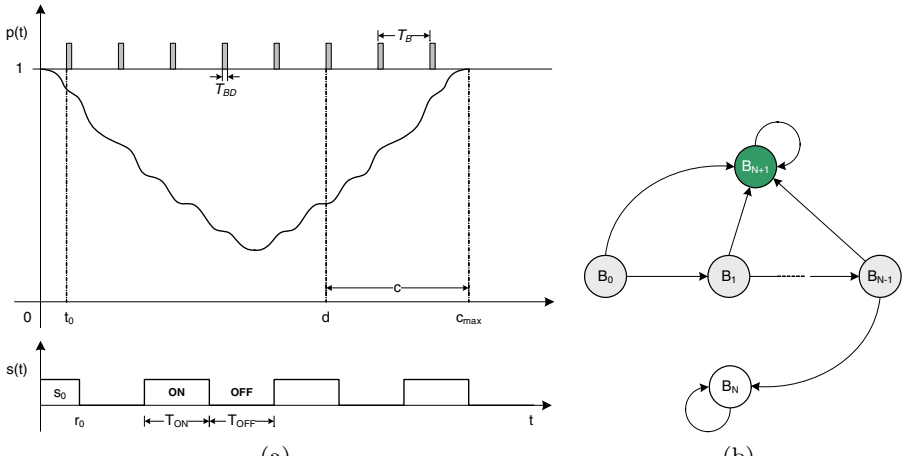
Similarly, the MR assumes that the communication is over when it does not receive any more message in a given period of time.

4 Discovery Phase Analysis

In this section we develop an analytical model for the discovery phase. The purpose of the analysis is to derive the distribution of the discovery time and, thus, the residual contact time as well. The analysis is split in two main parts. First, the state of the static node (i.e. ON or OFF) over time is derived, by keeping in consideration the duty cycle. Second, the beacon reception process is modeled, i.e. the state transitions of the static node are characterized, on the basis of the probability that a beacon sent by the MR at a given instant will be correctly received by the static sensor.

With the help of Fig. 2(a) we introduce the framework for the subsequent analysis. As beacon transmissions do not depend on when the MR enters the contact area, the initial beacon transmission within the contact area is generally affected by a random offset (with respect to the beginning of the contact time). In detail, the time instant at which the MR transmits the first beacon while in the contact area is denoted as t_0. As beacon transmissions are periodic and start at t_0, the actual instants of subsequent beacon transmissions can be expressed as $t_n = t_0 + n \cdot T_B$, with $n \in [1, N-1]$ where $N = \lceil c_{max}/T_B \rceil$ is the maximum number of beacons the MR can send while in the contact area. Therefore, if the MR is discovered by means of the m-th beacon, the discovery time is $d = d_m(t_0) = t_0 + m \cdot T_B$, and the corresponding residual contact time is $c = c_m(t_0) = c_{max} - d = c_{max} - d_m(t_0)$.

The state of the static node at a given instant is completely specified by its composite state (s, r) where s denotes the radio state, i.e. ON or OFF, and



(a) (b)

Fig. 2. Beacon discovery process (a) and beacon state (b)

r represents the residual time, i.e. the amount of time the node will remain in the same state s. The initial state of the static node at the time $t = 0$ is referred to as (s_0, r_0). Let us denote by $s(t)$ and $r(t)$ the radio state and the residual time, respectively, at a generic time t. Because of the duty cycle, both $s(t)$ and $r(t)$ evolve in a deterministic way. In detail, the radio state of the static node is periodic, with period equal $T_{ON} + T_{OFF}$. We focus now on the radio state $s(t_n)$ of the static node at beacon transmission times (t_n). As $s(t)$ is periodic, it is sufficient to investigate the remainder of the ratio between the beacon transmission times and the period of the duty cycle. By comparing this remainder against the initial residual state s_0 and the initial residual time r_0, it is possible to derive $s(t_n)$. Specifically, it is

$$\underset{s_0=\text{ON}}{s(t_n)} = \begin{cases} \text{ON} & \text{if } 0 \le t'_n < r_0 \\ \text{OFF} & \text{if } r_0 \le t'_n < r_0 + T_{OFF} \\ \text{ON} & \text{if } r_0 + T_{OFF} \le t'_n < T_{ON} + T_{OFF} \end{cases} \tag{1}$$

$$\underset{s_0=\text{OFF}}{s(t_n)} = \begin{cases} \text{OFF} & \text{if } 0 \le t'_n < r_0 \\ \text{ON} & \text{if } r_0 \le t'_n < r_0 + T_{ON} \\ \text{OFF} & \text{if } r_0 + T_{ON} \le t'_n < T_{ON} + T_{OFF} \end{cases} \tag{2}$$

where $t'_n = t_n \mod (T_{ON} + T_{OFF})$. Similarly, we can also derive the residual time $r(t_n)$

$$\underset{s_0=\text{ON}}{r(t_n)} = \begin{cases} r_0 - t'_n & \text{if } 0 \le t'_n < r_0 \\ T_{OFF} + r_0 - t'_n & \text{if } r_0 \le t'_n < r_0 + T_{OFF} \\ T_{ON} + T_{OFF} + r_0 - t'_n & \text{if } r_0 + T_{OFF} \le t'_n < T_{ON} + T_{OFF} \end{cases} \tag{3}$$

$$\underset{s_0=\text{OFF}}{r(t_n)} = \begin{cases} r_0 - t'_n & \text{if } 0 \le t'_n < r_0 \\ T_{ON} + r_0 - t'_n & \text{if } r_0 \le t'_n < r_0 + T_{ON} \\ T_{ON} + T_{OFF} + r_0 - t'_n & \text{if } r_0 + T_{ON} \le t'_n < T_{ON} + T_{OFF} \end{cases} \tag{4}$$

Once the duty-cycled state of the static node has been fully characterized, we have to model the actual beacon reception process. To this end we introduce the state representation illustrated in Fig. 2(b), where the states B_i, $i \in [0, N+1]$, refer to the static node at beacon transmission times. In detail, B_0 is the initial state in which the static node is waiting for the MR to transmit the first beacon in the contact area. B_j is entered after missing the first j beacons sent by the MR, where $j \in [1, N-1]$. B_N is entered when the static node has not detected the MR presence at all, because it has not received any of the beacons. Finally, B_{N+1} is entered when the static node has successfully received a beacon. When it is in a state B_k, with $k \in [0, N-1]$, the static node can only move to the state B_{k+1} or to the state B_{N+1} if it has lost or got a beacon, respectively. Note that B_N and B_{N+1} are absorbing states. In addition, the state of the static node is completely specified by its current state.

Now, we can derive a joint characterization of the radio state of the static node and the beacon reception process. For simplicity, time has been discretized in slots with duration Δ, so that the whole process can be modeled as a discrete time

Markov chain. For the sake of clarity, in the following we will not explicitly refer to time-dependent parameters by their actual discretized values, unless otherwise specified. The transition matrix H corresponding to the beacon reception process can be thus written as follows

$$\mathbf{H} = \begin{pmatrix} 0 & H_{01} & 0 & \cdots & 0 & H_{0,N+1} \\ 0 & 0 & H_{12} & & 0 & H_{1,N+1} \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & H_{N-1,N} & H_{N-1,N+1} \\ 0 & 0 & 0 & \cdots & H_{NN} & 0 \\ 0 & 0 & 0 & \cdots & 0 & H_{N+1,N+1} \end{pmatrix}$$

where the H_{kl} are sub-blocks denoting the transition probability from the state B_k to the state B_l. Note that the state B_0 is evaluated at time $t = 0$, while state B_i with $i \in [1, N]$ is evaluated at the i-th beacon transmission time, i.e. t_i. In addition to the state B related to the beacon reception, the H_{kl} blocks also keep track of the radio state of the static node. In detail, the elements of the H_{kl} block can be expressed as $h^{kl}_{(s_i,r_i),(s_j,r_j)} = \mathbb{P}\{B_l, (s_j, r_j) \,|\, B_k, (s_i, r_i)\}$. Since the state of the static node is deterministic, the only admissible transitions are those specified by the state equations (1-4), i.e. between the generic state (s_i, r_i) and the corresponding state (s_j^*, r_j^*) such that $s_j^* = s(t_k)$ and $r_j^* = r(t_k)$. Specifically, the transition probabilities are as follows

$$h^{kl}_{(s_i,r_i),(s_j^*,r_j^*)} = \begin{cases} 1 & \text{if } s_j^* = \text{OFF and } B_l \neq B_{N+1} \\ 0 & \text{if } s_j^* = \text{OFF and } B_l = B_{N+1} \\ p(t_k) & \text{if } s_j^* = \text{ON and } B_l \neq B_{N+1} \\ 1 - p(t_k) & \text{if } s_j^* = \text{ON and } B_l = B_{N+1} \end{cases}$$

The above probabilities can be justified as follows, assuming to be in the state B_k.

- If the radio will be OFF during the next beacon transmission (at time t_k), then the static node will miss the beacon for sure, so that it can only enter a state different from B_{N+1} (i.e. $B_l = B_{k+1}$).
- Otherwise, the static node will be active during the next beacon transmission time. The static node will miss the beacon with a probability $p(t_k)$, thus moving to the state $B_l = B_{k+1}$. Conversely, it will correctly receive the beacon with a probability $1 - p(t_k)$ thus entering the state $B_l = B_{N+1}$.

Let $\mathbf{X}^{(0)}$ be the initial state probability vector of the static node and $\mathbf{X}^{(k)}$ the state probability vector associated to the time of the k-th beacon transmission, with $k \in [1, N-1]$,

$$\mathbf{X}^{(k)} = \begin{pmatrix} X_0^{(k)} & X_1^{(k)} & \cdots & X_{N-1}^{(k)} & X_N^{(k)} & X_{N+1}^{(k)} \end{pmatrix}$$

$$\mathbf{X}^{(0)} = \begin{pmatrix} X_0^{(0)} & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

where only the $X_0^{(0)}$ component of the initial state vector is not zero, as when the MR enters the contact area the static node is waiting for the first beacon to be sent. By definition of discrete time Markov chain, it follows that

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} \cdot \mathbf{H} \quad \text{for } k = 0, 1, 2, \ldots, N-1 \tag{5}$$

Note that the X_{N+1}^k component of the state vector represents the cumulative probability of the MR discovery after k beacon transmissions. Hence, the p.m.f. of the discovery time r.v. D, i.e. $d(m, t_0) = \mathbb{P}\{D(t) = m\}$, can be derived as

$$d(m, t_0) = \begin{cases} X_{N+1}^{(0)} & \text{if } m = t_0 \\ X_{N+1}^{(k)} - X_{N+1}^{(k-1)} & \text{if } m = t_k,\ k \in [1, N-1] \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

To derive the p.m.f. of the discovery time by (5) we need to know the initial state probabilities for $\mathbf{X}^{(0)}$. As both beacon transmissions and activations of the static node are periodic and independent, it is reasonable to assume that the initial radio state and the initial residual time are uniformly distributed along all possible values. Hence, for both radio states and independent from the residual time, the initial probability is $\Delta/(T_{ON} + T_{OFF})$, where Δ is duration of a discretized time slot.

All the above discussion assumes a certain initial beacon transmission time t_0. To properly characterize the discovery time, the Equation (6) must be evaluated for all possible values of t_0. Again, as both beacon transmissions and activations of the static node are periodic and independent, we will assume that all possible values of t_0 are uniformly distributed in the range $0 \leq t_0 < T_B$. Note that t_0 has been discretized into $\hat{t}_0 \in \mathcal{T} \equiv \{0, \Delta, \ldots, n_{t_0} \cdot \Delta\}$, where $n_{t_0} = \lfloor T_B/\Delta \rfloor$ is the maximum number of discretized time slots Δ which fit into $[0, T_B)$. Hence, the p.m.f. $d(m)$ of the discovery time per contact is

$$d(m) = \sum_{\hat{t}_0 \in \mathcal{T}} d(m, \hat{t}_0) \cdot \mathbb{P}\{\hat{t}_0\} = \frac{\Delta}{T_B} \sum_{\hat{t}_0 \in \mathcal{T}} d(m, \hat{t}_0)$$

5 Data Transfer Phase Analysis

In this section we derive the amount of messages correctly transferred by the static node to the MR. Recall that the static node enters the data transfer phase after a successful beacon reception. Since this depends on the discovery time, we will make use of the p.m.f. $d(m)$ obtained in the previous section to derive the number of correctly transferred data messages.

As anticipated in Section 4, while in the data transfer state, the static sensor is always on, and uses an ARQ-based communication protocol for data transfer. In the following, we will assume that both data and acknowledgments messages have a fixed duration T_s, referred to as *message slot*. In addition, we will assume a window size of w messages.

We focus now on a single window starting at the generic time t. As the message loss changes with the distance between the MR and the static sensor, every message will experience its own loss probability. However, we will assume that the message loss is constant during a message slot, i.e. that the i-th message in the window starting at time t will experience a message loss probability $p(t + i \cdot T_s)$. This is reasonable, given the short duration of the message slot. Let's denote by $N(i,t)$ the r.v. denoting the number of messages successfully received by the MR in a given slot i of the window starting at time t. Clearly, the p.m.f. of $N(i,t)$ is $n(i,t,m) = \mathbb{P}\{N(i,t) = m\}$, i.e.

$$n(i,t,m) = \begin{cases} 1 - p(t + i \cdot T_s) & \text{if } m = 1 \\ p(t + i \cdot T_s) & \text{if } m = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Hence, the total number of messages received by the MR within a window, i.e. $N(t)$, is the sum of the $N(i,t)$ r.v.s $N(t) = \sum_{i=0}^{w-1} N(i,t)$, so that its p.m.f. is the convolution of the single p.m.f.s, i.e. $n(t,m) = \otimes_{i=0}^{w-1} n(i,t,m)$.

Furthermore, we denote by $R(t)$ the r.v. representing the number of messages correctly transferred to the MR when an ARQ-based mechanism is used. So, in this case $R(t)$ represents the number of messages acknowledged by the MR. In the following, we will consider a selective retransmission scheme, where acknowledgements notify the sensor node which messages sent in the last window have been correctly received by the MR. Hence, the reception of the acknowledgement has to be accounted as well, so that the messages within a window are correctly transferred if they are successfully received by the MR and the corresponding acknowledgement is not lost. We denote by $A(t)$ the r.v. indicating the number of acknowledgements correctly received by the MR for the corresponding window starting at time t. Hence, the p.m.f. of $A(t)$ is $a(t,m) = \mathbb{P}\{A(t) = m\}$, i.e.

$$a(t,m) = \begin{cases} 1 - p(t + w \cdot T_s) & \text{if } m = 1 \\ p(t + w \cdot T_s) & \text{if } m = 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

hence $R(t) = N(t) \cdot A(t)$ and, being them indepedent, we have that

$$\mathbb{E}[R(t)] = \mathbb{E}[N(t)] \cdot \mathbb{E}[A(t)] = \sum_{i=0}^{w-1} [1 - p(t + i \cdot T_s)] \cdot [1 - p(t + w \cdot T_s)]$$

5.1 Joint Discovery and Data Transfer

The above discussion focuses on a single communication window. To get the number of messages transferred during the whole contact time, we have to characterize both the starting time t of the first window and the total number W of windows actually available in the residual contact time. Hence, the total number of messages correctly transferred during a contact is

$$R = \sum_{i=0}^{W} R(t + i \cdot (w+1) \cdot T_s)|_{t=D} \tag{9}$$

If D is the r.v. denoting the discovery time, whose p.m.f. has been derived in the previous section, clearly the start time of the first communication window is $t = D$. In addition, the number of windows in the residual contact time is $W = \lfloor (c_{max} - D)/((w+1) \cdot T_s) \rfloor$, under the assumption that the static node can exploit all the residual contact time for data transfer. We verified by simulation that with an appropriate setting of N_{ack} it is possible to exploit the residual contact time almost completely (see Section 6.2). Thus, the number of messages transferred successfully can be expressed as the r.v. R which depends only on the discovery time D.

Energy Efficiency. In this section we derive the total energy consumed by the static node per message successfully delivered to the MR, on the average. This metric provides an indication of the energy efficiency for the overall data collection process. The energy consumed in a given radio state is calculated as $P_{rs} \cdot T_{rs}$, where P_{rs} is the power drained in the state rs while T_{rs} is the time spent in the same state. As possible radio states we consider rx for receive, tx for transmit and sl for sleep (i.e. shutdown). In addition, we assume that the power consumption when the radio is idle (i.e. it is monitoring the channel) is the same as in the receive state. As the energy efficiency depends on both discovery and data transfer, we derive first the energy consumption for the discovery phase, and then its joint effect on the subsequent data transfer.

Since the MR arrival may be unknown a priori, the static node may spend a waiting time σ in addition to the discovery time (see Fig. 1(a)). Hence, the energy spent during the discovery phase is

$$\overline{E}_{disc} = P_{sl} \cdot (\sigma + \mathbb{E}[D]) \cdot (1 - \delta) + P_{rx} \cdot (\sigma + \mathbb{E}[D]) \cdot \delta$$

where the first term accounts for the energy spent in the sleep state, while the second one accounts for the time spent in the active state before the correct reception of the beacon message.

On the other side, the energy spent for data transfer – under the assumptions that the static sensor has always data to send and data transfer takes the entire residual contact time – is

$$\overline{E}_{dt} = \left(\frac{\mathbb{E}[c_{max} - D]}{w+1} + \mathbb{P}\{D\} \cdot \frac{N_{ack}}{2} \cdot T_s \right) \cdot (w \cdot P_{tx} + P_{rx})$$

The first part of the equation denotes the number of windows in the contact time plus the windows wasted after the end of the contact time. Note that the wasted windows have to be considered only when the contact actually occurs, so that the related term is multiplied by the probability of correct detection of the MR (i.e. $\mathbb{P}\{D\} = X_N^{(N-1)}$). The second term, instead, denotes the amount of power spent for a single window in the receive and transmit states, respectively.

Finally, the average total energy consumed by the static sensor per each message correctly transferred to the MR is $\overline{E}_{msg} = (\overline{E}_{disc} + \overline{E}_{dt})/\mathbb{E}[R]$.

6 Results

In this section we will use the analytical formulas derived in the previous sections to perform an integrated performance analysis of the overall data collection process. To this end, we will consider the following performance metrics.

- *Residual contact ratio*, defined as the ratio between the average residual contact time and the contact time $\eta = \mathbb{E}\left[(c_{max} - D)/c_{max}\right]$.
- *Contact miss ratio*, defined as the fraction of MR passages not detected by the static sensor (i.e. $\mathbb{P}\{N\} = X_{N+1}^{(N-1)}$).
- *Throughput*, defined as the average number of messages (or bytes) correctly transferred to the MR at each contact (i.e. $\mathbb{E}[R]$).
- *Energy consumption per byte*, defined as the mean energy spent by the static sensor per each message (or byte) correctly transferred to the MR (i.e. $\overline{E}_{msg}$ as defined in Subsection 5.1).

In our analysis we used a message loss function derived from the experimental data presented in [13] and measured in the same scenario introduced in Section 3. To get a more flexible model, we considered an interpolated polynomial packet loss function in the form

$$p(t) = a_2 \cdot \left(t - \frac{c_{max}}{2}\right)^2 + a_1 \cdot \left(t - \frac{c_{max}}{2}\right) + a_0 \tag{10}$$

Equation (10) holds only within the contact area, i.e. for $0 < t < c_{max}$. For other values of t, $p(t)$ is assumed to be equal to one, as outside of the contact area any transmitted message is lost. To derive the coefficients in (10) – reported in Table 1(a) for different MR speeds v and for a vertical distance $D_y = 15$ m – we used the same methodology described in [12].

We evaluated the model derived in Sections 4 and 5 and validated the analytical results with a discrete event simulator written in C. In the following, we will show both analytical and simulation results. However, unless stated otherwise, we will refer to the analytical results. Table 1(b) shows the parameter settings for both analysis and simulation.

Table 1. Interpolated message loss coefficients as functions of the MR speed (D_y = 15m (a) and other parameters used for analysis (b)

(a)

Coefficient	$v = 3.6$ km/h	$v = 40$ km/h
a_0	0.133	0.4492
$a_1(\mathrm{m}^{-1})$	0	0
$a_2(\mathrm{m}^{-2})$	0.000138	0.0077

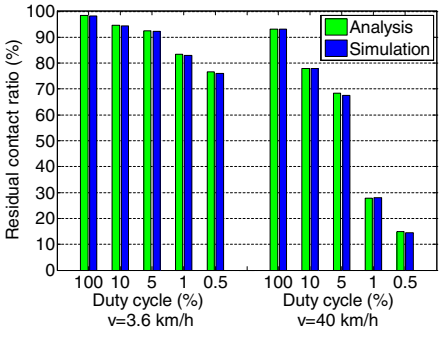
(b)

Parameter	**Value**
Transmit power (0 dBm)	49.5 mW
Receive (idle) power	28.8 mW
Sleep power	0.6 μW
Message payload size	24 bytes
Message slot size	15 ms
T_B	100 ms
T_{BD}	9.3 ms

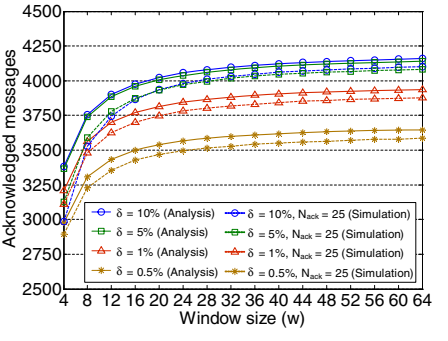
6.1 Discovery Phase

In this section we evaluate the performance of the discovery protocol, in terms of residual contact ratio and missed contacts. First of all, the effectiveness of the discovery protocol strictly depends on the T_B and T_{BD} parameters. As a design criterion, both parameters should be reduced as much as possible. In fact, a low T_B increases the frequency of beacon transmission, hence the probability of a timely beacon discovery. On the other hand, a low T_{BD} reduces the overhead due to the beacon reception, so that the residual contact time is not significantly decreased. The proper setting of both parameters strictly depends on the actual system on which the data collection protocols are implemented. In the following discussion we set $T_B = 100$ ms and $T_{BD} = 9.3$ ms, which have been found as suitable values for a Mote class sensor platform [14].

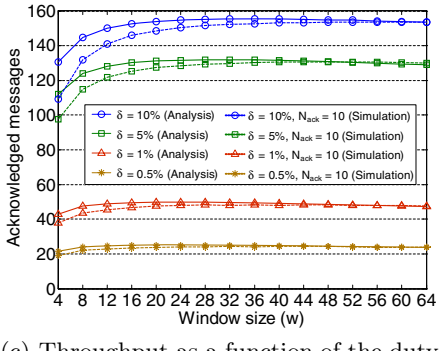
Figure 3(a) shows the residual contact ratio as a function of the MR speed, for different duty cycles. We start considering the MR moving at 3.6 km/h. The results clearly show that the duty cycle used for discovery does not significantly impact the residual contact ratio. For comparison purposes, we have also



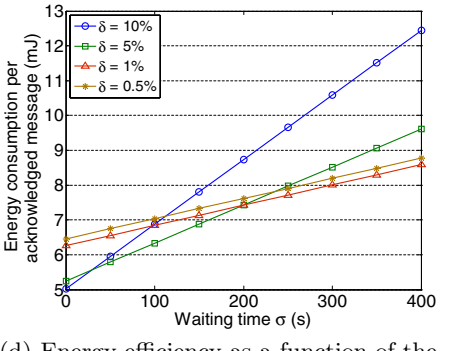
(a) Residual contact ratio



(b) Throughput as a function of the duty cycle for $v = 3.6$ km/h



(c) Throughput as a function of the duty cycle for $v = 40$ km/h



(d) Energy efficiency as a function of the waiting time for $v = 40$ km/h

Fig. 3. Analysis vs simulation results

included the scenario in which the static sensor is always active during discovery. In this case, the residual contact ratio is very close to the maximum achievable value. Actually, the static sensor can exploit most of the contact time also when it uses a moderate duty cycle. Also lower duty cycles can get satisfactory residual contact ratios, over the 75% for duty cycles of 1% and 0.5%, respectively. These results clearly depend on the average contact time – in this case equal to 158 s – which is much greater than the sleep time of the static node, so that the contact miss ratio is almost zero (i.e. 0.04%) even at a 0.5% duty cycle.

When the MR moves at 40 km/h we can see that the static node can still use a large part of the contact time – which is approximately 17 s – when it is always on during discovery, as the obtained residual contact ratio is above the 90%. However, the residual contact ratio drops when an even moderate duty cycle is used. For lower duty cycles (i.e. 1% and 0.5%), the residual contact ratio decreases significantly. In these cases, in fact, not only the residual contact ratio is much shorter, but there is also a high contact miss probability. This is because the 1% and the 0.5% duty cycles have a sleep time comparable to the contact time. As a consequence, the chance that the static node does not detect the passage of the MR at all is much higher than in the other cases.

6.2 Data Transfer

In this section we evaluate the performance of of an ARQ-based transfer protocol in terms of throughput. It is worth recalling that the following results have been obtained for the actual residual contact time, so that they account for the effects of the discovery phase as well.

Figure 3(b) and Figure 3(c) show the throughput obtained by analysis and simulation when the MR moves at different speeds. Note that analysis assumes that the data transfer phase takes all the residual contact time, while simulation uses the loss of N_{ack} consecutive acknowedgements as end-of-contact indication. This is the reason for the slight difference between analytical and simulation results. Actually, this difference is higher for low values of the window size. This happens because the time after which the sensor assumes the MR as out of reach is $(w+1) \cdot N_{ack} \cdot T_s$, so that it increases with the window size when N_{ack} is constant. When the window size is small, the actual value of N_{ack} should be increased in order to keep $N_{ack} \cdot (w+1)$ of the same magnitude. Anyway, the value of N_{ack} should be tailored to the target scenario, which depends on the speed of the MR. For instance, in our simulation we used $N_{ack} = 25$ and $N_{ack} = 10$ for the 3.6 km/h and the 40 km/h scenarios, respectively.

We start considering the throughput in terms of messages acknowledged by the MR. Figure 3(b) shows the throughput as a function of the window size when $v = 3.6$ km/h. We can see that the throughput increases with the window size for all considered duty cycles, so that the maximum is obtained with the largest window size of 64 messages. This is due to the acknowledgement overhead, which decreases as the window size grows up. Specifically, the throughput reaches over 4000 messages per contact, corresponding to about 100 kB of data, for the 10% duty cycle. Similar results are also obtained with the lower 5% and 1% duty

cycles. Such results can be explained on the basis of the residual contact ratios, which are similar for the different duty cycles, i.e. the residual contact time is not reduced significantly when a low duty cycle is used. The same is not true for the 0.5% duty cycle, which actually experiences a lower, but still reasonable, throughput of about 3500 messages per contact (nearly 88 kB).

Figure 3(c) shows the throughput as a function of the window size when $v = 40$ km/h. The throughput has a different trend in this case. It first increases when the window size is low, then decreases after a point which depends on the duty cycle. In addition, the obtained throughput changes significantly with the duty cycle. In fact, while the 10% and the 5% duty cycles both achieve a similar throughput over 100 messages per contact (around 3 kB), the 1% duty cycle gets only 50 messages (1.2 kB) per contact. The lowest 0.5% duty cycle even obtains a throughput of 25 messages per contact (0.6 kB), which is rather low, but may be enough for certain applications.

6.3 Energy Efficiency

In this section we evaluate the energy efficiency of data collection. It should be noted that the considered energy consumption accounts for both discovery and data transfer, so that it fully characterizes the overall data collection process. While in the previous sections we have considered only what happens within the contact area, in the following we take into account also the time spent by the static sensor on waiting for the MR to enter the communication range. To this end, we measured the average energy spent per acknowledged message as a function of the waiting time.

Figure 3(d) shows the average energy consumption per acknowledged message when the MR moves at $v = 40$ km/h and the window size is 32. Clearly the energy consumption increases with the waiting time, but a very low duty cycle is not necessarily the most convenient option. In fact, when the average waiting time is below 25 s (i.e. the MR arrival can be predicted with rather good accuracy), the best option is the 10% duty cycle. Instead, when the average waiting time is between 25 s and 200 s, the most convenient duty cycle is 5%. From later on, i.e. when the MR presence is very difficult to estimate, the best duty cycle is 1%. In addition, the 0.5% duty cycle always gets a higher average energy consumption than the 1% duty cycle. These results are in contrast to the expected behavior, i.e. that the energy consumption decreases with the duty cycle, as it happens in the scenario where the MR moves at 3.6 km/h (we have omitted the correspondent figure for the sake of space).

Actually, these results can be explained as follows. Low duty cycles may delay the MR detection, leading to lower residual contact times. In addition, they may also produce high contact miss ratios, so that the energy spent during discovery is simply wasted, as the sensor node cannot transmit any data. In the considered scenario, where the MR moves with $v = 40$ km/h, the contact time and the residual contact time are short. When the waiting time is low, i.e. when the sensor knows the MR arrival times with a good accuracy, the energy overhead due to discovery is negligible, because the sensor spends most of its active time

during data transfer. Instead, the advantages of low duty cycles become relevant when the waiting time is high. In this case, the sensor may spend a significant amount of time by looking for beacons when the MR is out of the contact area. Thus, a low duty cycle reduces the activity of the sensor during the waiting time, which is the highest share of the overall energy consumption. Regarding the 0.5% duty cycle, it is always unsuitable in this scenario, because the energy gain due to the lower activity time is thwarted by the decrease in the throughput (see Fig. 3(c)).

7 Conclusions

In this paper we have developed an analytical model of the overall data collection process in sparse sensor networks with mobile relays (MRs). The model is flexible enough to incorporate different discovery and data transfer protocols. We limited our discussion to a simple discovery algorithm where the MR sends periodic advertisements and the sensors follow an asynchronous scheme based on a low duty cycle. In addition, we considered an ARQ communication protocol with selective retransmission for data transfer. Our findings show that low duty cycles can be actually used for a large class of environmental monitoring applications. Surprisingly, a low duty cycle may not always be the most energy efficient option, depending on a number of different factors such as the speed and the mobility of the MR.

This work could be improved along different directions. First, the model proposed in this paper could be extended to the case of multiple MRs. Second, different discovery and data transfer schemes could be considered. Finally, the findings of our analysis could be used as a basis for the definition of adaptive data collection protocols, which are capable to tailor the operating parameters to the actual conditions (i.e. knowledge of the MR arrivals, buffer constraints, residual energy of sensor nodes etc.). We are currently evaluating these extensions as a future work.

Acknowledgements

Work funded partially by the European Commission under the FP6-2005-NEST-PATH MEMORY project, and partially by the Italian Ministry for Education and Scientific Research (MIUR) under the FIRB ArtDeco project. The authors would like to thank Emmanuele Monaldi, Francesca Mancini and Paolo Caggiari for their help.

References

1. Jain, S., Shah, R., Brunette, W., Borriello, G., Roy, S.: Exploiting mobility for energy efficient data collection in wireless sensor networks. ACM/Springer Mobile Networks and Applications 11(3), 327–339 (2006)
2. Chakrabarti, A., Sabharwal, A., Aazhang, B.: Using predictable observer mobility for power efficient design of sensor networks. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 129–145. Springer, Heidelberg (2003)

3. Shah, R.C., Roy, S., Jain, S., Brunette, W.: Data mules: Modeling a three-tier architecture for sparse sensor networks. In: Proc. IEEE SNPA 2003 (2003)
4. Jea, D., Somasundara, A., Srivastava, M.B.: Multiple controlled mobile elements (Data mules) for data collection in sensor networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 244–257. Springer, Heidelberg (2005)
5. Kansal, A., Somasundara, A., Jea, D., Srivastava, M., Estrin, D.: Intelligent fluid infrastructure for embedded networks. In: Proc. ACM Mobisys 2004 (2004)
6. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: A survey. In: Ad Hoc Networks (in press)
7. Zhao, W., Ammar, M.: Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In: Proc. IEEE FTDCS 2003 (May 2005)
8. Jun, H., Ammar, M., Zegura, E.: Power management in delay tolerant networks: A framework and knowledge-based mechanisms. In: Proc. IEEE SeCon 2005 (2005)
9. Gu, Y., Bozdag, D., Ekici, E.: Mobile element based differentiated message delivery in wireless sensor networks. In: Proc. IEEE WoWMoM 2006 (2006)
10. Gu, Y., Bozdag, D., Ekici, E., Ozguner, F., Lee, C.: Partitioning based mobile element scheduling in wireless sensor networks. In: Proc. IEEE SECON 2005, pp. 386–395 (2005)
11. Somasundara, A., Kansal, A., Jea, D., Estrin, D., Srivastava, M.: Controllably mobile infrastructure for low energy embedded networks. IEEE Transactions on Mobile Computing 5(8) (2006)
12. Anastasi, G., Conti, M., Monaldi, E., Passarella, A.: An adaptive data-transfer protocol for sensor networks with Data Mules. In: Proc. of IEEE WoWMoM 2007 (2007)
13. Anastasi, G., Conti, M., Gregori, E., Spagoni, C., Valente, G.: Motes sensor networks in dynamic scenarios. International Journal of Ubiquitous Computing and Intelligence 1(1) (April 2007)
14. Anastasi, G., Conti, M., Di Francesco, M.: Data collection in sensor networks with Data Mules: an integrated simulation analysis. In: Proc. of IEEE ISCC 2008 (2008)

MVSink: Incrementally Building In-Network Aggregation Trees

Leonardo L. Fernandes[1,2] and Amy L. Murphy[2]

[1] University of Trento
[2] Fondazione Bruno Kessler – IRST
Trento, Italy
{leiria,murphy}@fbk.eu

Abstract. In-network data aggregation is widely recognized as an acceptable means to reduce the amount of transmitted data without adversely affecting the quality of the results. To date, most aggregation protocols assume that data from localized regions is correlated, thus they tend to identify aggregation points within these regions. Our work, instead, targets systems where the data sources are largely independent, and over time, the sink requests different combinations of data sources. The combinations are essentially aggregation functions. This problem is significantly different from the localized one because the functions are initially known only by the sink, and the data sources to be combined may be located in any part of the network, not necessarily near one another. This paper describes MVSink, a protocol that lowers the network cost by incrementally pushing the aggregation function as close to the sources as possible, aggregating early the raw data. Our results show between 20% and 30% savings over a simplistic approach in large networks, and demonstrate that a data request needs to be active only for a reasonably short period of time to overcome the cost of identifying the aggregation tree.

1 Introduction

In-network data aggregation is rapidly becoming the accepted mechanism to reduce the amount of transmitted data in a wireless sensor network without significant loss of data quality [1]. This requires both the selection of the aggregation function, a highly application-dependent task, as well as the identification of the *best* node at which to apply the function. Most existing approaches assume that data from a localized region can be fused together, thus they focus on identifying one or more nodes in each region, rotating the aggregation function evaluation among them.

While many applications fit this scenario, we are motivated by a different class of applications in which the sensors are more heterogeneous, and neither the required data nor the aggregation function are known in advance. An aggregation function is a function that combines data from different sources. Aggregation functions can be as simple as an average of many readings or more sophisticated

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 216–231, 2009.
© Springer-Verlag Berlin Heidelberg 2009

functions that take data from different types of sensors and make some application specific decision, for example, combining data from smoke detectors and temperature sensors to infer if there is a fire. We identified this problem during our prior work on the MiLAN middleware [2], which, in summary, takes as input a large set of sensors, and over time selects different subsets that, when combined according to user-defined functions, meet a minimum quality constraint. This choice of the subset and its duration of use is made to maximize the total system lifetime. In MiLAN, we assume that the functions operating on data are applied at the sink, but in this work we recognize that moving the functions into the network reduces the amount of information that needs to be transmitted, thus increasing system lifetime.

Our overall goal, therefore, is to identify the nodes where the aggregation function should be applied, building a cost-effective aggregation tree. For this, we take an incremental, in-network approach, assuming that the sink node is initially the only node with knowledge of the function. The function is then incrementally *pushed* farther into the network until the *best* locations are found, forming an aggregation tree. The novelty presented in this paper is a set of heuristics that recognize when the function should be split, with distinct parts moving toward different sets of sources. The optimal aggregation solution is a Steiner tree that includes all the sources and the sink. Finding a Steiner tree in an arbitrary graph is known to be NP-hard [3] even for centralized algorithms. Therefore, our completely distributed approach aims to identify an approximation of the optimal Steiner tree using novel heuristics to incrementally identify better locations for the aggregate functions. Nevertheless, this approximation is an improvement over a solution that applies all functions at the sink or that applies a shortest paths tree (SPT), as detailed in our evaluation.

Section 2 describes MVSink, our novel protocol for incrementally moving an aggregation function from the sink closer to the data sources that provide its input. The section emphases the heuristics used for pushing the function to increasingly cost effective locations. Our results, presented in Section 3, indicate that our approach provides significant benefit at reasonable cost. Section 4 places this work in the context of related efforts while Section 5 ends the paper with brief concluding remarks.

2 MVSink

This section begins with brief descriptions of the system model and basic definitions upon which our description of MVSink relies. We then overview the protocol operation, then detail four heuristics we use for identifying the lowest cost aggregation tree.

2.1 System Model

We assume a standard network environment in which nodes are connected with bidirectional links. Any unidirectional links are pruned by a standard MAC

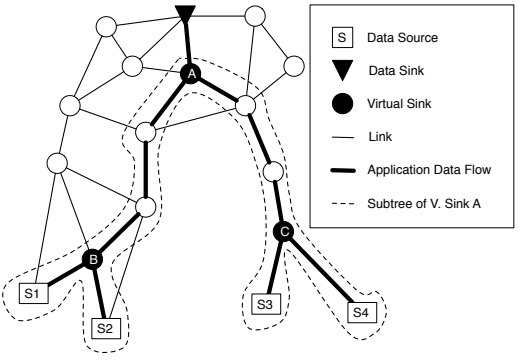


Fig. 1. Example of a network with nodes acting as sources, sink and virtual sinks

protocol. We further assume that all nodes operate in promiscuous mode, overhearing transmissions by their one-hop neighbors, whether or not the packet is destined for them.

2.2 Basic Definitions

We begin with some basic definitions illustrated in Figure 1 and used throughout the paper. How these components combine in a coherent protocol is described next.

- *Aggregation Tree:* We represent the sensor network itself as an undirected, connected graph $G(V, E)$ with a single sink node and a set of data sources $S \subset V$. An aggregation tree is a connected subgraph of G containing the sink, the sources S and any other nodes and edges needed to connect these nodes without creating cycles. Our protocol seeks to minimize the size of this tree. The bold edges in Figure 1 identify the edges of the minimal aggregation tree for this network.
- *Virtual Sink:* A virtual sink is a node in the aggregation tree that receives data from two or more sources and applies an aggregation function, shrinking the amount of data forwarded to the sink. Figure 1 shows three virtual sinks, A, B, and C .
- *Candidate Node:* A candidate node is a *candidate* for serving as a virtual sink, meaning it has the potential to aggregate a subset of the sources currently aggregated by a virtual sink. All candidates are within k hops of a current virtual sink, where k is a tunable parameter.
- *Subtree:* The subtree of a virtual sink or candidate node n is the tree that contains n as a root and all the nodes and edges that compose the paths between each source n aggregates (or proposes to aggregate) and itself. Figure 1 highlights the subtree of virtual sink A.

2.3 Protocol Operation

In a nutshell, MVSink seeks to minimize the size, in number of edges, of the aggregation tree by incrementally moving the virtual sink from its initial location at the data sink to a location closer to the sources. If necessary, the virtual sink can be split into multiple pieces, creating some sinks that continue to migrate deeper in the aggregation tree, closer to the source, while leaving one virtual sink behind to combine the data from these deeper virtual sinks.

The protocol executes the following steps repeatedly, until no better aggregation tree can be found:

1. The virtual sink announces in broadcast to its k-hop neighbors its id and the set S of data sources it currently aggregates. k is a protocol parameter;
2. Nodes receiving such an announcement and that have overheard data flowing from two or more nodes in S identify themselves as virtual sink candidates as they can aggregate data from some set of sources. This candidacy is communicated to the current virtual sink along with the information about the subset of S that can be aggregated and the hopcount from the candidate to each of these nodes. Intuitively, we include individual hop counts as it allows the virtual sink to select a candidate based on the cost for any set of sources it can aggregate. Sections 2.5 through 2.6 provide further details.
3. After receiving the candidacy messages, the virtual sink decides, based on one of the heuristics we define later, which of the candidates should assume the role of virtual sink and start aggregating data. There are multiple options. For example, a single candidate can be chosen to aggregate all the sources in S. Alternately, the virtual sink can be split among multiple candidates, having each of them aggregate a subset of S. If no candidates propose a cost/effective solution or if there are no candidates, the protocol ends with the current virtual sink.
4. Any newly assigned virtual sinks start this process from the beginning.

The primary contribution of this paper is the proposal of four novel heuristics applicable in step 2 described above. The following sections outline each.

2.4 Largest Set Heuristic

Intuitively, the more data a virtual sink aggregates, the more effectively it can reduce the amount of data flowing in the network. Therefore, our first, and most straightforward heuristic, simply identifies the candidate node with the largest set of proposed sources. If multiple candidates can aggregate the same number of sources, the choice is arbitrary or the best local gain heuristic, discussed next, can be applied as a tie-break.

After the first candidate is selected and the sources it can cover are removed from S, S may not be empty. The suitability of the remaining candidates is evaluated to cover these sources, identifying additional virtual sinks. This first heuristic employs very small candidacy messages, containing only the set of sources proposed to be aggregated by the candidate.

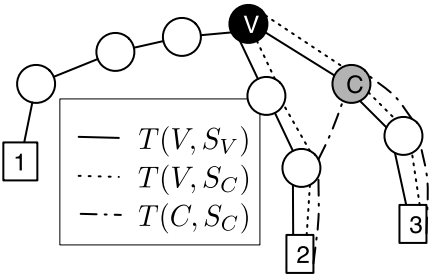


Fig. 2. Representation of the terms in (1). The set of sources for virtual sink V is $S_V = \{1, 2, 3\}$ and for candidate C is $S_C = \{2, 3\}$.

2.5 Best Local Gain Heuristic

Given that our goal is to reduce the size of the aggregation tree, our next heuristic selects candidate virtual sinks based on the size of the subtree they yield. For example, in Figure 2, the candidate node C proposes to aggregate sources $S_C = \{2, 3\}$ with a subtree size of 4. As the cost for the current virtual sink to aggregate this set of sources is 6, this represents an improvement in the overall size of the aggregation tree of 1, as the link between V and C must be considered. Formally, if $T(X, S_Y)$ is the number of edges in the subtree that connects a node X to the sources node Y aggregates, S_Y, where $X = Y$ is allowed, then the cost if candidate C is selected by virtual sink V is given by:

$$Cost(C) = T(V, S_V) - T(V, S_C) + T(C, S_C) + d(V, C) \quad (1)$$

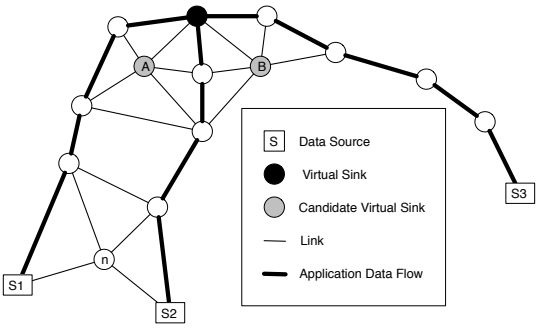
where $d(V, C)$ is the distance from V to C, in this case 1. Figure 2 shows the T costs. This formula is applied to each candidate, and the one with the lowest $Cost(C)$ is chosen. In case of a tie, the node farthest from the virtual sink (i.e., with the largest $d(V, C)$) is selected, with the goal of moving the aggregation as close to the sources as possible in a single step.

If the selected candidate does not aggregate all the sources aggregated by V, the remaining sinks and candidates are considered, always selecting the highest *Cost* value until either there are no more sources, or there are no more suitable candidates.

This heuristic can also be used as a tiebreak strategy for other heuristics.

2.6 Affinity Based Heuristics

We developed two affinity based heuristics in order to avoid the local minima trees that are occasionally found in the previous heuristics. Such local minima occur due to the lack of global information available at the virtual sink. A virtual sink, with access to information from its neighborhood only, cannot identify candidates potential to make progress with subsequent virtual sinks selection closer to the data sources.



Affinity Matrix Kept by Virtual Sink

	S1	S2	S3
S1	-	5	0
S2	5	-	3
S3	0	3	-

Fig. 3. Example use of affinities. With only local (1-hop neighborhood) information, the virtual sink cannot decide whether it is better to select candidate A or candidate B as the next virtual sink. Candidate A proposes to aggregate data from $S1$ and $S2$, whereas candidate B can aggregate sources $S2$ and $S3$. In the previous heuristics, candidates A and B are equivalent choices, but the figure clearly shows that choosing candidate B would result in a dead end, since there would be no further suitable candidates in the network to aggregate $S2$ and $S3$, while candidate A is a much better choice, since the protocol would then be able to proceed until reaching node n as a virtual sink.

Figure 3 shows the motivation for the next two heuristics. In the picture, there are two virtual sink candidates. Candidate A can aggregate sources $\{S1, S2\}$, and candidate B proposes to aggregate $\{S2, S3\}$. From the figure, it is clear that sources $S1$ and $S2$ are more closely related than $S2$ and $S3$, making candidate A the best choice. However, relying only on local (1-hop) information, the virtual sink cannot identify the best option. We refer to this relationship between $S1$ and $S2$ as *affinity*, and develop a heuristic to identify it.

We define the affinity between two given sources as a numeric value kept by every node i, reflecting approximately the distance from node i to a potential aggregation point. In the figure, node n is a potential aggregation point for $\{S1, S2\}$ and is farther from the virtual sink than node B, which is the farthest possible aggregation point for $\{S2, S3\}$. Therefore, the affinity perceived by the virtual sink is higher for $\{S1, S2\}$ than it is for $\{S2, S3\}$.

Affinity information is introduced in the network by nodes that overhear messages from two or more sources, a situation which indicates their ability to serve as a virtual sink. Unfortunately, if such a node is not transmitting data, this affinity information will not reach the current virtual sink. However, because the node has already overheard application messages, a single, one-hop broadcast of this affinity information is enough to reach a node that *is* transmitting data. This broadcast, referred to as an affinity message, carries affinity information that is then forwarded with application messages toward the sink.

Affinity information is kept and transmitted as an $n \times n$ matrix, where n is the number of sources. If all sources are not known in advance, a data structure

containing only the information about known sources can be used. To save memory and reduce packet size, a triangular matrix can be used instead of a full matrix, since the values in the upper and lower parts of the matrix are equivalent (e. g. $M_{a,b} = M_{b,a}$).

Each node keeps the affinity values of all pairs of sources it knows, initially all affinity values are zero. When a node overhears messages from two sources with affinity zero between them, it sets the affinity to 1 and sends its affinity matrix to its neighbors. The neighboring nodes update their local matrices, which are transmitted with subsequent application messages. When a node receives an affinity matrix $A = (a_{i,j})_{n \times n}$ in an application message, the local matrix $B = (b_{i,j})_{n \times n}$ updates all of its entries as follows:

$$b_{i,j} = \begin{cases} \max(b_{i,j}, a_{i,j} + 1), & \text{if } (a_{i,j} > 0) \\ b_{i,j}, & \text{otherwise.} \end{cases} \quad (2)$$

Since application messages are always forwarded through the current aggregation tree, virtual sinks receive updated affinity information. Since the positive values are incremented at each hop of application messages, the potential candidates that are farther from the virtual sink receive higher priority, as desired.

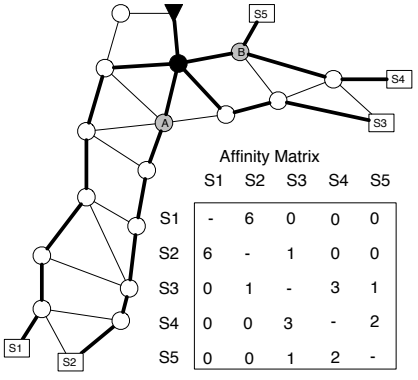
The broadcast of affinity messages by nodes that are not already transmitting data represents an extra cost to MVSink. Although affinity messages are limited to a single one-hop broadcast per node, the number of such nodes can be large, increasing the total cost of the protocol. This cost is discussed in the evaluation section, where the benefits it provides are clearly shown to outweigh the cost.

Next, we briefly describe two heuristics that rely on affinity information to choose between virtual sink candidates.

Best Affinity Candidate Heuristic. This heuristic simply selects the candidate that proposes to aggregate the set of sources with largest affinity. When candidates propose to aggregate more than two sources, the affinity value of the set is the average of the affinity values of each individual pair of sources in the set. When affinity values are equal, the best local gain heuristic can be used as a tiebreak.

Best Affinity Set Heuristic. This heuristic is similar to the best affinity candidate heuristic, but it tries to redistribute the sources among the candidates in a way that increases the total affinity in the sets rather than simply choosing between the sets proposed by the candidates.

This strategy is motivated by situations in which candidates propose to aggregate a large set S which effectively "steal" sources that have more affinity with sources that are not part of S. For example, in Figure 4(a) a virtual sink aggregates five sources $S1$ through $S5$. $S1$ and $S2$ have affinity 6. $S3$ and $S4$ also have a good affinity value of 3. Now suppose there are two candidates A and B proposing to be virtual sinks. Node A proposes to aggregate $S1$, $S2$ and $S3$ and node B can aggregate $S3$, $S4$ and $S5$. The previous heuristic would let node A aggregate its three sources, even though there is little affinity between $S3$ and the other sources in A's set. A better solution would be to let A aggregate $S1$



Affinity Matrix

	S1	S2	S3	S4	S5
S1	-	6	0	0	0
S2	6	-	1	0	0
S3	0	1	-	3	1
S4	0	0	3	-	2
S5	0	0	1	2	-

(a) Example virtual sink and its affinity matrix.

Pair	Set of Sets	Feasibility
$S1, S2$	$\{\{S1, S2\}\}$	$A\{S1, S2\}$
$S3, S4$	$\{\{S1, S2\},$ $\{S3, S4\}\}$	$A\{S2, S1\}$ $B\{S3, S4\}$
$S3, S2$	{{S1,S2,S3,S4}}	Not feasible, undo.
$S3, S5$	$\{\{S1, S2\},$ $\{S3, S4, S5\}\}$	$A\{S2, S1\}$ $B\{S3, S4, S5\}$

(b) Best affinity set heuristic execution. Pairs added in decreasing order of affinity and the resulting subsets according to algorithm 1.

Fig. 4. Motivating example for the best affinity set heuristic. Note that according to the protocol there would be more candidates in this example, but for simplicity we consider only candidates A and B.

and $S2$, and node B aggregate $S3$, $S4$ and $S5$, to take advantage of both good affinity pairs.

Instead of considering the candidate sets, the virtual sink, using its local affinity matrix, incrementally builds the best affinity sets and tries to find candidates that can aggregate them.

Algorithm 1, executed on each virtual sink, outlines the heuristic. Essentially, the algorithm adds each pair of sources, in decreasing order of affinity, to a set of sets S to be assigned to virtual sink candidates. Adding a pair of sources to S means that both sources should be in the same set of S, since there is good affinity. The operation can result in adding both sources to a set in S or in the merging of two sets of S, as detailed in algorithm 2. Figure 4 shows an example execution, with the pairs of sources added and the resulting set of sets generated at each step. Sets are built in order to maximize the total affinity between all sets. After the addition of each pair, the algorithm tests if there are candidates that can cover S, adding only pairs that produce sets feasible to be distributed among the available candidates.

The `feasible` procedure returns a boolean value of true if there are candidates available that can cover the sets in `doneSources`.

3 Evaluation

Intuitively the movement of the virtual sinks closer to the sources lowers network cost, and thus has the potential to improve system lifetime. This section supports this intuition with an evaluation through simulation showing both costs and benefits. Two settings were used for simulations. Random deployment of sensors

Algorithm 1. The best affinity set heuristic

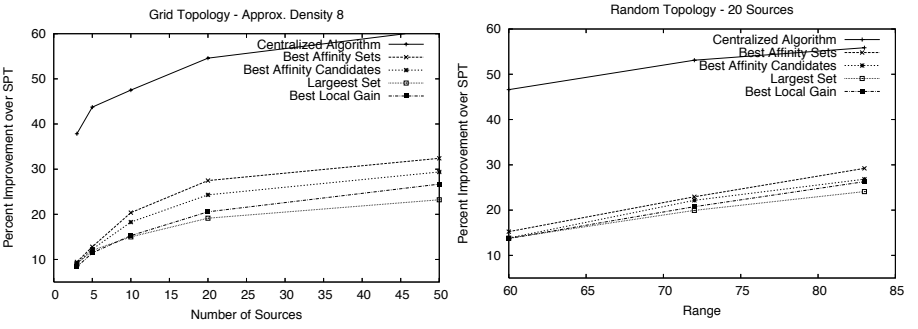
```
Set mySources = aggregatedSources
SetOfSets doneSources = ∅
Matrix aff = affinityMatrix
while |mySources| > 1 and hasNextHigher(aff) do
   pair (a,b) = getNextHigher(aff)
   doneSources = addPair(a,b,doneSources)
   remove(a,b,mySources)
   if !feasible(doneSources) then
      undo last addPair and remove calls
   end if
end while
Distribute sets to resp. candidates
```

Algorithm 2. The addPair function.

```
Require: setOfSets and sources a and b as parameters
  if setOfSets contains both sources a and b then
     merge sets of a and b in setOfSets
  else if setOfSets contains source a then
     add b to the set that contains a
  else if setOfSets contains source b then
     add a to the set that contains b
  else {setOfSets does not contain a or b}
     add new set {a, b} to setOfSets
  end if
  return setOfSets
```

and distribution of sensors in a grid topology. The main difference between the two topologies is the uniformity of grids. In random deployments there are often connectivity holes and antiholes (subgraphs in which every vertex is adjacent to every other) [4]. The presence of holes may result in local minima for virtual sink placement. Antiholes can increase the cost of protocol operation due to broadcasting in high density areas. With grids, node connectivity is uniform throughout the network, avoiding both holes and antiholes. In both topologies, data sources are randomly chosen among the nodes and data sinks are placed at a corner of the simulated area. In the random topology, the deployment area is a 1000×1000 square. In the grid topology, 1000 nodes are arranged in a 40×25 grid with nodes uniformly distributed. In all simulations, we use $k = 2$ for the virtual sink announcements. We assume a perfect aggregation function (e.g. average), meaning that a packet of aggregated data has the same size as a raw data packet.

We vary the communication range, the number of data sources and the number of nodes in our simulations in order to evaluate the performance of MVSink in different network sizes and densities. We compare MVSink's different heuristics: largest set, best local gain (BLG), best affinity candidate and best affinity set; against shortest paths trees (SPT) and against a centralized algorithm.



(a) Variable number of sources (b) Variable range

Fig. 5. Percent improvement of the heuristics over the shortest path tree

The shortest paths tree, or opportunistic aggregation tree, is formed by each source sending application messages to the sink along a shortest path between the two. Overlapping shortest paths are combined to form the aggregation tree.

The centralized approach is based on the nearest participant first algorithm by Takahashi and Matsuyama [5]. The heuristic starts with the tree containing only the sink. In each step, the closest source to the current tree is connected. This process is repeated until all sources are connected. The output of this heuristic is an approximation to the optimal Steiner tree containing all the sources and the sink. The performance ratio of the algorithm is 2, in the worst case. Although the centralized approach is significantly better than MVSink, it requires global information which is not readily available and is very costly to obtain.

Each graph represents an average of 250 simulation runs. Error bars show standard deviation. The large values for standard deviation are due to results being largely dependent on the placement of data sources in the network. Networks with most sources close to the sink or to each other present results significantly different from cases with most sources far away from the source and sparsely distributed.

Since the main focus of this work is on the algorithm behavior, we used Sinalgo, a simulator for network algorithms [6], abstracting away from low-level network concerns. We assume reliable, constant delay, bidirectional channels, and while we acknowledge that limitations exist in real deployments, they will not affect the fundamental correctness of MVSink. We have also run simulations with up to 10% message loss (except for the sink transferring messages, which must be reliably delivered) and obtained results only slightly worse than those presented here.

3.1 Improvement over Shortest Paths Tree

Since SPT is a common way of performing aggregation in sensor networks, in this section we compare the percent improvement over SPT of the different heuristics

and the centralized approach. The comparison allows us to clearly identify the performance differences, in terms of tree size, between the four proposed heuristics. The percent improvement is equivalent to the reduction in the tree size of each heuristic compared to SPT.

Figure 5(a) shows this comparison in a grid with a density of 8 neighbors per node, except at the edges. As expected, the best affinity sets approach is the best of our heuristics, due to its characteristic of selecting better aggregation sets at each step and its capability to collect information from nodes closer to the sources. Best affinity sets trees were close to 9% smaller than SPT with 3 sources and approximately 30% better for the case with 50 sources. The best local gain heuristic obtained better results than the largest sets approach, however it is still inferior to both affinity based heuristics.

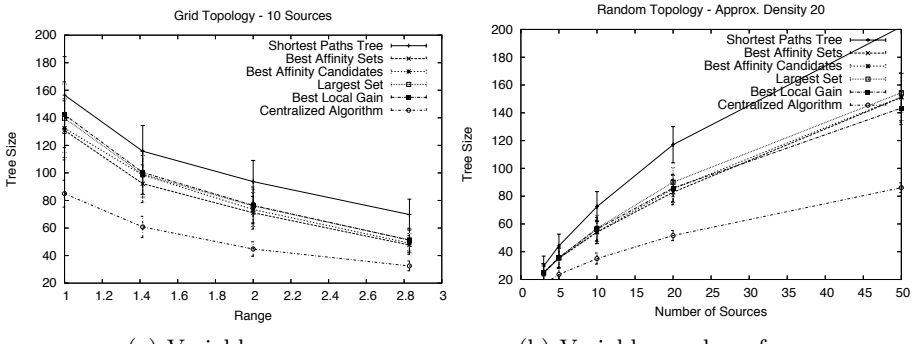
In Figure 5(b) a similar graph is shown for a random topology and variable communication range yielding densities between 10 and 20 neighbors per node. Here we observe the improvement of the protocols as density grows. In this scenario, the affinity based heuristics are superior to the other two.

Our results also lead us to conclude that the protocol works better in the grid topology than in the random one. The results for the grid with density 8 and 20 sources in Figure 5(a) are better than for the random topology with 20 sources and density 10, the leftmost points in Figure 5(b). This difference was expected due to the regularity of grids.

3.2 Reducing Transmission Cost

The aggregation tree size is equivalent to the number of transmissions it takes to transmit data from all of the sources to the sink. In this sense, minimizing the tree size is crucial for energy savings and increasing network lifetime.

The graph in Figure 6(a) shows the comparison between our heuristics, the centralized approach, and SPT. There is a clear tendency for MVSink to approach the centralized algorithm as communication range grows. This improvement is due to the virtual sink announcements reaching more nodes in higher



(a) Variable range

(b) Variable number of sources

Fig. 6. Tree sizes of MVSink heuristics, shortest paths tree and centralized approach

density scenarios. There are also more possible candidates, since more nodes can overhear application messages in dense networks. The larger groups of candidates to choose from allows MVSink to make better decisions and to get closer to the quality of the centralized algorithm.

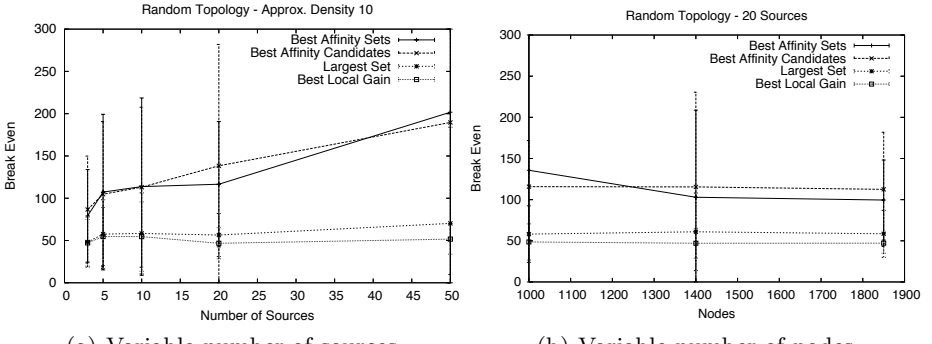
Figure 6(b) demonstrates the consistent advantage of MVSink over the opportunistic aggregation scheme for different numbers of sources. MVSink tree sizes are significantly better than SPT in all cases. The protocol scales well for all ranges of sources simulated. Figure 6(b) shows, for example, that for runs with fifty data sources MVSink on average achieves a tree size with approximately 140 edges, while SPT in the same scenario builds trees with size over 200.

We note a clear advantage of the affinity based heuristics over both the BLG and the largest set heuristics. As expected, the extra information collected provides the affinity oriented virtual sinks the possibility to make better decisions. But the extra information imposes higher costs. The affinity messages transmission cost is significant and must be taken into consideration. In the next section we discuss the costs of each heuristic and provide a cost/benefit analysis.

3.3 Overhead Evaluation

We measure the cost of MVSink as the number of transmissions it takes to: perform 2-hop broadcasts of all virtual sink announcements, send all unicast candidacy and virtual sink assignment messages, and, in the case of the affinity based heuristics, send all affinity messages. The cost is shown as the number of times that the final aggregation tree must be used until the message savings (compared to SPT) compensate for the costs of MVSink. We call this value the break even point. For this analysis we set the cost of MVSink messages equal to that of application messages. In many systems, application messages would be larger than control messages and in such cases, the gains of using MVSink would pay off even earlier.

In Figure 7(a), we can see that the break even is quite stable as the number of sources increases. The affinity based heuristics get more expensive with more



(a) Variable number of sources (b) Variable number of nodes

Fig. 7. Protocol Overhead

sources, but still obtain a quite good average break even value of less than 200 for the considerable number of 50 data sources. For the BLG and largest sets heuristics, the situation is even better, since the break even does not grow as the number of sources increases. The small number of extra virtual sink announcements, assignment and candidacy messages it takes those heuristics to deal with larger numbers of sources (i.e. larger trees) are completely compensated by the extra aggregation gain of smaller trees discussed in Sections 3.1 and 3.2.

Figure 7(b) shows the break even as a function of the network density. To achieve increasing density, we simulated runs with different numbers of nodes in the same area. The picture shows that as density grows, the break even costs remain stable for all heuristics. In higher density scenarios, an increasing number of messages is necessary to run the protocol. Denser 2-hop broadcast announcements and the consequently larger number of candidacy messages, for instance, affect the costs significantly. The number of affinity messages is also heavily influenced by density, since more potential virtual sinks are likely to be found. Despite all the extra cost required to run the protocol on higher density networks, the aggregation gains improve enough to keep the break even costs reasonably stable for all heuristics.

In both break even graphs it is clear that the affinity based heuristics are more expensive than the other two, due to the need for affinity messages. These results, combined with the observation from the previous sections that the affinity based heuristics provide better trees, indicate that the affinity heuristics are more suitable for longer lived queries, i.e. that use the tree for longer periods, while BLG and largest sets heuristics are more appropriate for building trees that will be used a smaller number of times.

4 Related Work

Aggregation has been extensively studied in the literature, and in general has been shown to provide significant performance gains in a wide range of scenarios [1].

Some aggregation oriented protocols require additional information to be known in advance or transmitted throughout the network for a large cost. Greedy Incremental Tree (GIT) [7,1], for example, incrementally builds an aggregation tree by first selecting the shortest path connecting the sink to the closest source and then connecting other sources to the tree, one at a time, at the closest point between of the current tree. However, to do so GIT requires that "exploratory samples have initially and repeatedly been flooded throughout the network" [7]. GIT uses the large amount of global information this makes available, but the flooding is costly, even prohibitively so on large networks. Another approach, by Kansal et. al. [8] requires node location information, which is often unavailable in sensor networks. The work by Ramachandran et.al. [9] requires specific data fusion roles to be known in advance and needs some topological information for achieving a good initial placement of such fusion points. Our protocol does not require any additional knowledge above what is provided by a simple

broadcast sink announcement to establish routes, such as that provided in standard versions of directed diffusion [10] and many other protocols.

Other proposed solutions [11,12,13] build approximations to a minimum spanning tree (MST) to be used as an aggregation tree. Minimum spanning trees are good solutions for the case in which all nodes are sources, or when using the same structure to deal with any arbitrary query. MVSink, instead, deals with a different problem: approximating the optimal tree for a specific set of sources on a per-query basis. The optimal solution in the case of MVSink is a Steiner tree containing the sink and all the sources. Also, it is often the case in sensor networks that the radio transmission radius is not tunable, meaning that essentially all transmissions have the same cost (i.e. all edges have the same weight). In such cases, any arbitrary spanning tree is an MST. Therefore, we claim that an arbitrary spanning tree is not necessarily a good solution.

Other protocols provide aggregation in hierarchical topologies [14,15,16]. Nodes are divided in clusters with special nodes, cluster heads, that aggregate the data from all nodes in the cluster. Such solutions rely on a hierarchy from which generating an aggregation tree is straightforward. Often cluster based approaches also rely on the capability of cluster heads to transmit data directly to the sink [14,17], which is very costly, especially on large networks. Gao et. al. [18] propose a protocol for sparse data aggregation that forms an aggregation forest, each tree being connected to a node in the boundary of the network. The approach assumes nodes in the boundaries of the network have special capabilities to communicate directly with the data sink. Fan et. al. propose structure-free data aggregation [19], based on data-aware anycast transmissions (e.g. receivers with data to aggregate have priority) and randomized waiting to achieve aggregation. The approach is only suited for applications with high time and spatial convergence, such as event based applications and does not provide aggregation for sparse data on either time or space. Also, the approach assumes the use of geographic information, which is often not available on sensor networks. Our approach does not make any topological assumptions. It does not rely on special nodes or on one-hop communication to the sink and is thus applicable in any random, connected topology. MVSink takes advantage of spatial convergence and can also find suitable solutions for reasonably sparse data sources.

Many works address aspects such as quality of service [20], security [21] or load balancing [8] and assume the a priori existence of a proper aggregation tree over which to apply their techniques. The focus of MVSink is on the more basic problem of finding a good tree. Therefore such techniques can be applied after MVSink generates the aggregation tree.

One of the main concerns regarding the use of aggregation is the latency it adds to the network. But, as shown by Zhu et al. [20], in extreme cases where there is too much traffic in the network, the use of aggregation can actually reduce latency in the network. It is important to notice that these are also the cases in which aggregation is most useful.

5 Conclusion

The work presented in this paper, the MVSink protocol, finds aggregation trees for applying a single aggregation function to an arbitrary number of sources in a wireless sensor network. While useful for a wide range of applications and aggregation functions, another scenario requires multiple, unique functions. We trivially solve this problem by running multiple instances of MVSink, one for each function. For example, to apply $F(G(S_1, S_2), H(S_3, S_4))$, where S_1 through S_4 are data sources and F, G and H are different aggregation functions, initially two instances of the protocol are started to solve G and H. Then, a third instance is started to solve F, considering the nodes that apply G and H as data sources.

In general, MVSink is unique in its approach to incrementally move aggregation points from the sink towards the sources, making it applicable in sink-driven data collection scenarios. This paper focused on four novel heuristics for virtual sink selection, and our evaluation showed the benefits of finding low-cost aggregation trees clearly outweighs the overhead of the protocol for moderately long-lived aggregation scenarios.

References

1. Krishnamachari, B., Estrin, D., Wicker, S.B.: The impact of data aggregation in wireless sensor networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), Washington, DC, USA, pp. 575–578. IEEE Computer Society Press, Los Alamitos (2002)
2. Heinzelman, W., Murphy, A.L., Carvalho, H., Perillo, M.: Middleware to support sensor network applications. IEEE Network Magazine Special Issue (2004)
3. Gröpl, C., Hougardy, S., Nierhoff, T., Prömel, H.J.: Lower bounds for approximation algorithms for the steiner tree problem. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 217–228. Springer, Heidelberg (2001)
4. Nikolopoulos, S.D., Palios, L.: Hole and antihole detection in graphs. In: SODA 2004: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 850–859. Society for Industrial and Applied Mathematics (2004)
5. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. Math. Japonica 24, 573–577 (1980)
6. Distributed Computing Group at ETH-Zürich: Sinalgo - simulator for network algorithms, http://dcg.ethz.ch/projects/sinalgo/
7. Intanagonwiwat, C., Estrin, D., Govindan, R., Heidemann, J.: Impact of network density on data aggregation in wireless sensor networks (2001)
8. Kansal, A., Srivastava, M.B.: An environmental energy harvesting framework for sensor networks. In: Proceedings of the International Symposium on Low power Electronics and Design (ISLPED), pp. 481–486. ACM Press, New York (2003)
9. Ramachandran, U., Kumar, R., Wolenetz, M., Cooper, B., Agarwalla, B., Shin, J., Hutto, P., Paul, A.: Dynamic data fusion for future sensor networks. ACM Transactions on Sensor Networks 2, 404–443 (2006)
10. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proceedings of the 6th International Conference on Mobile computing and Networking (MobiCom), pp. 56–67. ACM Press, New York (2000)

11. Ding, M., Cheng, X., Xue, G.: Aggregation tree construction in sensor networks. In: IEEE 58th Vehicular Technology Conference, VTC 2003-Fall, vol. 4, pp. 2168–2172 (2003)
12. Khan, M., Pandurangan, G., Vullikanti, A.: Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems (2008)
13. Cheng, H., Liu, Q., Jia, X.: Heuristic algorithms for real-time data aggregation in wireless sensor networks. In: IWCMC 2006: Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing, pp. 1123–1128. ACM, New York (2006)
14. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: An application-specific protocol architecture for wireless microsensor networks. IEEE Transactions on Wireless Communications 1(4), 660–670 (2002)
15. Manjeshwar, A., Agrawal, D.P.: TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In: Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS), Washington, DC, USA, p. 189. IEEE Computer Society Press, Los Alamitos (2001)
16. Wen, Y.F., Lin, F.Y.S.: Energy-efficient data aggregation routing and duty-cycle scheduling in cluster-based sensor networks. In: 4th IEEE Consumer Communications and Networking Conference, 2007, CCNC 2007, pp. 95–99 (2007)
17. Lindsey, S., Raghavendra, C.S.: Pegasis: Power-efficient gathering in sensor information systems. In: Aerospace Conference Proceedings, 2002, vol. 3, pp. 1125–1130. IEEE, Los Alamitos (2002)
18. Gao, J., Guibas, L., Milosavljevic, N., Hershberger, J.: Sparse data aggregation in sensor networks. In: IPSN 2007: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 430–439. ACM, New York (2007)
19. Fan, K.W., Liu, S., Sinha, P.: Structure-free data aggregation in sensor networks. IEEE Transactions on Mobile Computing 6, 929–942 (2007)
20. Zhu, J., Papavassiliou, S., Yang, J.: Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks. IEEE Transactions on Parallel and Distributed Systems 17, 923–933 (2006)
21. Chan, H., Perrig, A., Song, D.: Secure hierarchical in-network aggregation in sensor networks. In: Proceedings of the 13th ACM Conference on Computer and Communications security (CCS), pp. 278–287. ACM Press, New York (2006)

The Minimum Number of Sensors – Interpolation of Spatial Temperature Profiles in Chilled Transports

Reiner Jedermann and Walter Lang

Microsystems Center Bremen (MCB)
University of Bremen, Otto Hahn Allee NW1, D-28359 Bremen, Germany
rjedermann@imsas.uni-bremen.de

Abstract. Wireless sensor networks are an important tool for the supervision of cool chains. Previous research with a high number of measurement points revealed spatial temperature deviations of more than 5 °C in chilled transport, but the number of sensors has to be reduced to an economically useful value for use in regular transport. This paper presents a method to estimate the minimum number of sensors and to compare different sensor positioning strategies. Different methods of interpolating the temperature data of intermediate positions were applied to the experimental data from a delivery truck. The average prediction error for intermediate points was estimated as a function of the number of sensors. The Kriging method, originally developed for the interpolation of geostatistical data, produced the best results.

Keywords: Wireless sensor networks, Food logistics, Kriging, Information Processing, Temperature mapping.

1 Wireless Sensors in Cool Chain Management

Product losses in food transportation due to temperature mismanagement and quality decay can reach up to 35% [1]. These losses can be mitigated by better supervision of the cool chain. If stock rotation is based on dynamic shelf life or current product quality instead of a fixed production date, quality losses in meat could be reduced from 16% to 8%, as the group of Taoukis has shown [2], and those in fish could be reduced from 15% to 5% [3].

The prediction of shelf life requires temperature monitoring for individual product batches. Temperature differences inside a truck of up to 12 Kelvin [4] can cause severe deviations in product quality. Sea containers with air ducts in the floor allow for a better air flow distribution, but differences between pallet surfaces and core temperatures can still reach 6 Kelvin [5].

Wireless sensor networks provide online access to temperature data during transport. Instant notifications of food quality problems allow for corrective actions to be taken before the transport arrives at its destination. The prediction of shelf life losses can be calculated by an automated system inside the means of transport [6]. Sensors packed inside the cargo are often lost by the end of the transport. In order to avoid

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 232–246, 2009.
© Springer-Verlag Berlin Heidelberg 2009

high sensor replacement costs, it is not feasible to equip each box or pallet inside a truck or container with a sensor node. Instead of oversampling the cargo hold by implementing a high number of measurement points, the temperature has to be interpolated between the positions of a reduced number of sensors. Furthermore, the operating conditions of food transport place high physical demands on the sensors. The sensors must be able to operate in temperatures below -20 °C, withstand high air humidity and condensation, and endure cleaning by steam jet.

2 Required Number of Sensors

The goal of this study is to develop a method to the estimate the number of sensors required to accurately interpolate a spatial temperature profile. The difference between the real temperature and the temperature predicted by interpolation increases for low numbers of source or input sensors. Inappropriate positioning of the input sensors and inaccuracies of the interpolation methods also lead to higher errors.

This paper begins by defining a measure for the interpolation error and introducing the experimental data. The following section compares different interpolation methods for a fixed number of input sensors. A further analysis of the data tests to what extent the interpolation error is reduced by increasing the number of sensors. The effects of different strategies to determine locations for the placement of additional sensors are evaluated. The last section demonstrates how the interpolation error could also be utilized as an indicator of the probability of sensor faults.

2.1 Source and Destination Points

The measurement data set was split into two groups. The first group of sensor locations serves as the input for the interpolation model. This group contains the N_S source points s_i. The second group of sensor locations serves as a reference. The measurements at the N_Z destination points z_i were compared to the output of the selected interpolation model.[1]

2.2 Definition of Interpolation Error

The error ε_i for one destination point i was defined as the median square deviation between the predicted $\hat{z}_i$ and the measured z_i over N_K samples for the transport duration:

$$\varepsilon_i^2 = \frac{\sum_{k=1}^{N_K} \left(\hat{z}_i(k) - z_i(k)\right)^2}{N_K} \tag{1}$$

[1] Vectors are marked by bold lowercase letters, matrices by bold capital letters, and transposed matrices by an additional superscript 'T'. Temperature differences are given in Kelvin [K] and absolute temperatures are given in degrees Celsius [°C]. Averages are indicated by an overbar. Context clarifies whether the average is taken over the transport duration or over all measurement points for a certain sampling instance.

The quality of the interpolation methods was evaluated according to the average prediction error $\bar{\varepsilon}$ over all destination points:

$$\bar{\varepsilon} = \frac{\sum_{i=1}^{N_Z} \varepsilon_i}{N_Z} \tag{2}$$

3 Experimental Data

With the purpose of conforming to the mechanical requirements of cool chain transport, we equipped TelosB sensors with polyamide water-protected (IP65) housings and external SHT75 temperature and humidity sensors. In order to evaluate the error of the temperature interpolation, it is necessary to acquire data for more points than the final number of sensor positions. Because we have only manufactured a limited number of sensors, we performed the preliminary tests with low-cost data loggers. The tests were performed inside delivery trucks provided by the German company Rungis Express, which specializes in supplying high-quality food products to hotels and restaurants. The cargo hold of each truck is separated into three different temperature zones. During two test transports, 40 TurboTag data loggers were placed in each middle compartment in deep freezer mode with a set point of -29 °C.

Further details of the test are described in [7], whereas this paper focuses on an analysis of the experimental data to determine the minimum number of sensors and their optimal positions.

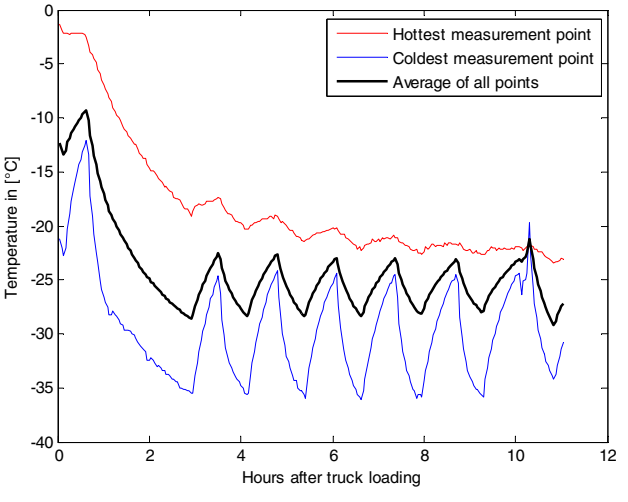


Fig. 1. Temperature over time. Minimum, maximum, and average for transport 1.

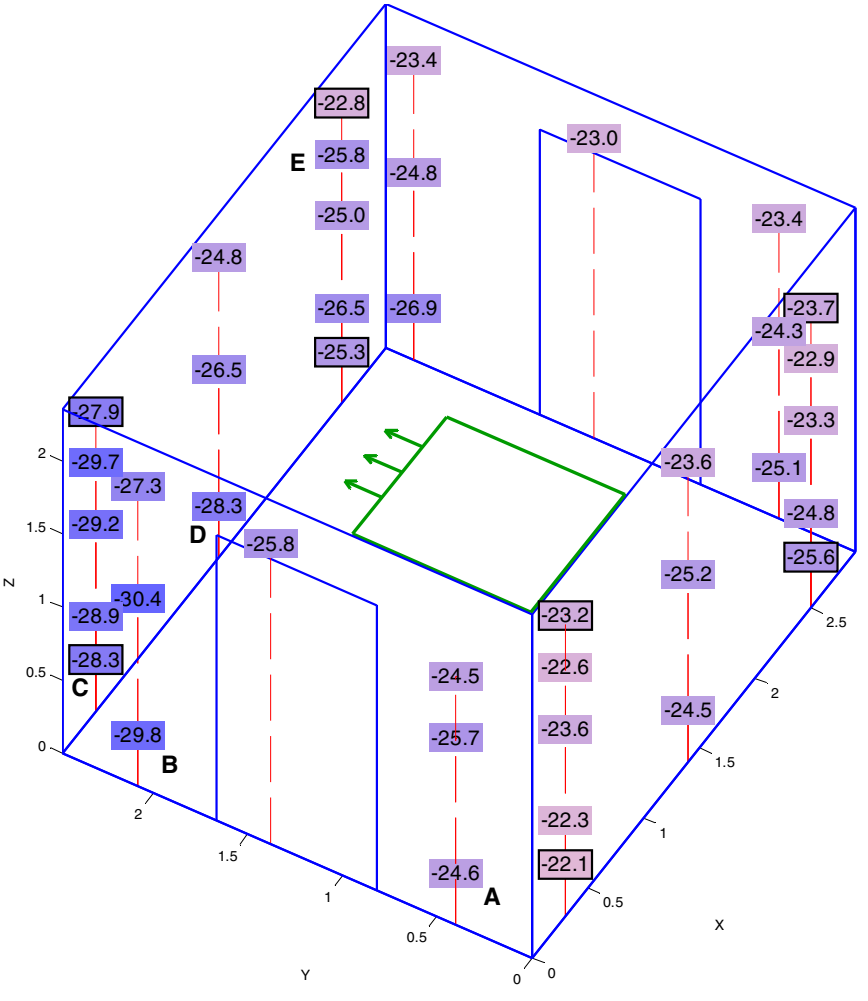


Fig. 2. Spatial temperature distribution at the end of the cooling period in transport 1. The cooling unit is mounted under the ceiling. The direction of air flow at the ventilation outlet is marked by the three arrows.

Figure 1 shows the temperature over time diagram of both the coldest and the hottest measurement points, as well as the average temperature of all points over the 11-hour cooling period during the first transport. The oscillations of the temperature were caused by the on/off cycles of the cooling unit with a period of approximately 1.2 hours. The automated defrosting caused a short temperature peak after 10.5 hours.

The spatial temperature distribution is given in Figure 2. The average over time was calculated for each measurement point over a two hour period starting 8 hours after truck was loaded. Temperature differences of about 7.5 K are still present at the end of the transport.

3.1 Time Correction

Data loggers are cheap and easy-to-use wireless devices, but they do not provide access to the sensor data during the transport. Furthermore, they do not feature networked time synchronization, which meshed sensor nodes provide. Data loggers can exhibit considerable deviations of the sampling intervals, especially in deep freezer conditions. We found clock deviations of ±3 % in our experimental data. The data was converted to an equal sampling interval of 2.5 minutes by the *resample()* function in MATLAB. This function resamples the input with a fixed frequency ratio of *p/q* by upsampling the input by the factor *p*, filtering alias frequencies, and downsampling by the factor *q*. A graphical comparison showed that the error introduced by the resampling was lower than 0.05 Kelvin except for occasional peaks of up to 0.1 Kelvin. Only samples at the beginning and end of the data set, which could have a higher error, were removed before further processing.

4 Methods for Spatial Interpolation

The following section compares different approaches for estimating the temperature at the destination points by a linear combination of the values at the source points. The prediction of the destination point $\hat{z}_i$ at the sampling instance *k* is given by the sum of the data of the source points s_j multiplied with the time-invariant weighting coefficients w_{ij}. The forecast depends solely on the current source values for these linear methods; their prediction model does not contain any state variables.

$$\hat{z}_i(k) = \sum_{j=1}^{N_S} s_j(k) \cdot w_{ij} \tag{3}$$

4.1 Inverse Distance Weighting

Inverse distance weighting is most common method for determining the weighting coefficients. This method uses only the geometrical distances between the source points. It assumes that the influence of a source point s_j on a destination point z_i decreases with the square of their distance h_{ij}. The weighting coefficients are given by equation (4). The additional parameters ω_i are used to scale the weighting coefficients in such a way that their sum equals 1 for each destination point.

$$w_{ij} = \frac{\omega_i}{h_{ij}^2} \text{ with } \omega_i = \left(\sum_{i=1}^{N_Z} \frac{1}{h_{ij}^2} \right)^{-1} \tag{4}$$

4.2 Kriging

An improved interpolation method was developed by D.G. Krige [8] in the 1950s for the exploration of mineral resources with 1000 or more test drillings. To date,

the Kriging method has only been scarcely utilized in sensor networks; see [9] for example. This might be due to the fact that, in general, there are a lower number of probe points in wireless sensor applications than in geological research. We tested whether Kriging could also calculate a precise prediction for small data sets with only 40 points of experimental data. The Kriging method, as described in equation (5)-(8) [10], [11], was implemented as a set of MATLAB functions. As new applications of Kriging we used this method to estimate the number of required sensors and to test the plausibility of the sensor data.

Kriging calculates the solution with the least possible expected value for the errors ε_i of each destination point by assuming the following:

a) The mean of the measurement values is independent of space, and
b) The expected value for the temperature difference between two points depends solely on their spatial distance vector.

Depending on the data set, in many cases it is possible to reduce the second assumption to an isotropic form in which the difference does not depend on the direction but only on the absolute value of the distance.

Kriging can be seen as an improved form of inverse distance weighting. Whereas the inverse distance method calculates the weighting coefficients directly by the geometrical distance between two points h_{ij}, Kriging uses the variogram $v(h_{ij})$ that expresses the statistical dependency of two points as function of their distance h_{ij}. The variogram describes the statistical dependency by the expected value E for the square of the temperature difference of two points i,j:

$$v(h_{ij}) = \frac{1}{2} E\left\{\left(s_i(k) - s_j(k)\right)^2\right\} \tag{5}$$

The primary disadvantage of the Kriging method lies in its estimation of the variogram from the measurement values. Because our data set with 40 sensor locations did not allow for a determination of separate variograms for different directions of the distance vector, an isotropic distribution was assumed. In contrast to geological research in which only one static value per probe point is taken, we obtained a time-variant series of measurements. The equation to estimate the experimental variogram $v^*(h)$ from the data was slightly modified, resulting in the following:

$$v^*_{i,j}\left(h_{i,j}\right) = \frac{1}{2} \cdot \sum_{k=1}^{N_K} \left(s_i(k) - s_j(k)\right)^2 \tag{6}$$

The value was calculated for the first transport with all $0.5 \cdot 40 \cdot (40\text{-}1) = 780$ possible combinations of two points. The resulting values for $v^*(h)$ were grouped by the absolute value of the distance in intervals of 0.25 meter length. The average value for $v^*(h)$ inside each interval is given by a marker in Figure 3.

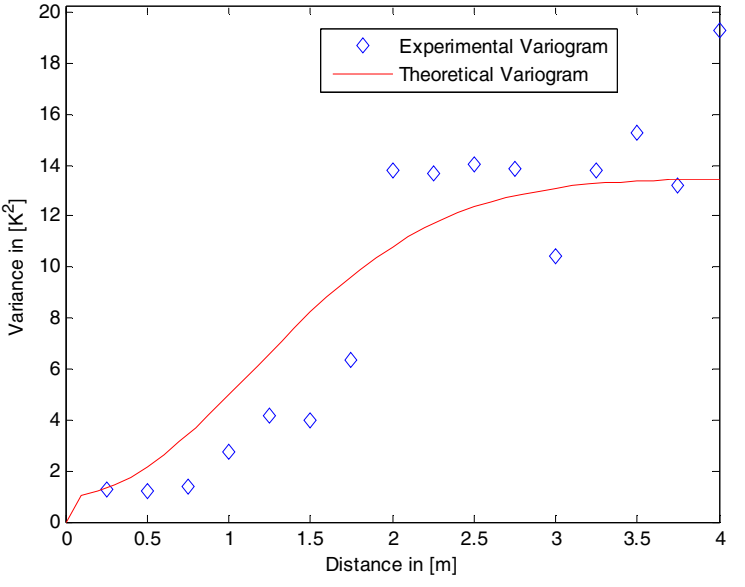


Fig. 3. Experimental and theoretical variogram

This experimental curve was approximated by a theoretical variogram *v(h)*. Only a limited set of functions can be applied as theoretical variograms. The function must conform to several limitations. For example, the variogram has to be a monotonically increasing function. In addition, apart from the origin, the function has to be continuous.[2] The Gaussian model was selected from the supposed standard models [10], [11] because it produced the best fit for the gentle rise of the curve for small distances:

$$v(h) = v_0 + (v_\infty - v_0) \cdot \left(1 - e^{-(3h/r)^2}\right) \tag{7}$$

The radius r as the primary parameter of the variogram can be interpreted as the maximum distance for the mutual influence between two points. The initial value v_0 is the minimum value of variance for distances greater than zero. v_∞ gives the variance for large distances. The parameters of the variogram for the data from the first transport were estimated to be $r = 2.8$ m, $v_0 = 1.0$ K², $v_\infty = 13.5$ K².

Analyses of the data from the second transport showed higher initial and final values ($v_0 = 2.5$ K², $v_\infty = 32.5$ K²), but no significant difference in the radius.

The method of inverse distance weighting directly calculates the weighting coefficients as a function of distance, whereas the Kriging method also considers the mutual influence of all measurement points by a linear system of equations. The Ordinary Kriging method also estimates the spatial average μ as an auxiliary variable:

[2] Only functions that are negative semi-definite can be utilized as variogram [11]. Otherwise, equation (12) on page 12 could result in a negative value for the Kriging variance.

$$\begin{bmatrix} v_{1,1} & \cdots & v_{1,N_S} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ v_{N_S,1} & \cdots & v_{N_S,N_S} & 1 \\ 1 & \cdots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_{1,q} \\ \vdots \\ w_{N_S,q} \\ \mu \end{bmatrix} = \begin{bmatrix} v_{1,q} \\ \vdots \\ v_{N_S,q} \\ 1 \end{bmatrix} \tag{8}$$

The matrix on the left side contains the values of the variogram for the distances between all source points $v_{i,j}$. The vector on the right side contains the values $v_{i,q}$ for the distances between each source point and one destination point z_q. The weighting coefficients $w_{i,q}$ used in calculating the destination point are retrieved by solving the linear set of equations in (8).

4.3 Kriging with Spatial Trends

The Ordinary Kriging method assumes that the mean value of the temperature is constant over space, but our experimental data showed a difference of 4 K between the average temperature of the left and right walls. There are three methods in which this problem can be addressed.

The first method is to simply ignore the trend and directly apply Ordinary Kriging, assuming that the effect of the trend is negligible for small distances.

The second method implements the following steps: a) A linear or polynomial trend model is estimated from the source points. b) The prediction of the trend model is subtracted from the source points. c) Ordinary Kriging is applied to the difference. d) The trend model is added to the destination points obtained by Ordinary Kriging.

The third approach using the Universal Kriging method extends the set of linear equations in (8). The parameter values of a trend model are calculated as additional variables. However, Universal Kriging is only necessary if the temperature distribution exhibits local drifts in addition to a global trend [10, page 38].

Because our data set of 40 measurement points does not contain enough information to estimate local drifts or cubic trend functions, only the second approach involving a linear model was applied. The temperature trend T^* was predicted as a function of the coordinates p_x, p_y, p_z, of a point i. The model parameters α were estimated separately for each sampling instance k:

$$T^*(i,k) = \alpha_0(k) + \alpha_1(k) \cdot p_x(i) + \alpha_2(k) \cdot p_y(i) + \alpha_3(k) \cdot p_z(i) \tag{9}$$

For the first transport, the average over time of the parameters was calculated as $\overline{\alpha}_0 = -21.98\,°\text{C}$, $\overline{\alpha}_1 = 0.17$ K/m, $\overline{\alpha}_2 = -1.98$ K/m, $\overline{\alpha}_3 = 0.15$ K/m.

5 Comparison of Interpolation Errors

The described interpolation methods were applied to the recorded data of both transports. We evaluated the data set twice, once using 8 source points and once using 30 source points. From the 40 total points, either 32 or 10 remained as destination points. The average interpolation error was calculated for these remaining points. The set with 8 source points includes the positions in the corners of the cargo hold that are

marked by a black frame in Figure 2. The positions of the sensors for the set with 30 source points were selected according to the approach described in the next section.

Two further simple interpolation approaches were applied as references for the comparison: a) The time-dependent average of the source points was taken to predict all destination points, independent of their locations. b) The destination values were set according to the trend model in equation (9). The results of the comparison are summarized in Table 1:

Table 1. Comparison of interpolation error $\bar{\varepsilon}$ for different methods for 8 and 30 source points

Experiment / Source points	Ex1 / 8	Ex2 / 8	Ex1 / 30	Ex2 / 30
Average of source points	2.796 K	3.912 K	2.567 K	3.239 K
Linear trend	1.984 K	2.723 K	1.881 K	3.437 K
Inverse distance weighting	1.443 K	2.287 K	1.105 K	1.720 K
Ordinary Kriging	1.389 K	2.170 K	0.530 K	1.325 K
Kriging with linear trend	1.418 K	2.231 K	0.533 K	1.474 K

Ordinary Kriging gave the most accurate prediction with an interpolation error between 0.5 K and 2.2 K. The combination of Kriging with a linear trend model resulted in a slightly higher error. A linear trend, which affects the whole cargo hold, fails to properly explain the difference between the average temperatures of the left and right walls. Thus, for this data set, the best approach is to directly apply Ordinary Kriging, which gives the best fit for randomly distributed heat sources.

The accuracy of Kriging increases with the number of source points. A comparison of the interpolation error of Ordinary Kriging with other methods gave the following results: For 30 source points, the error is reduced by 52% for the first and 23% for the second transport compared to inverse distance weighting. Compared to the simple average and the linear trend model, the improvement is between 59% and 79%.

The variograms for the first and second transports exhibited differences only in their initial and final values, with a similar relation of v_∞/v_0. The variogram for the second experiment can be approximated by a proportional scaling of the first variogram:

$$v_{\mathrm{sec}\,ond}(h) \approx 2.4 \cdot v_{first}(h) \tag{10}$$

The factor 2.4 can be reduced in equation (8) because it appears on both sides. Therefore, the weighting matrix for Kriging is almost independent of the number of the experiment. The effect of the differences in the variogram was calculated to be lower than 0.1 %.

5.1 Comparison with Linear Curve Fitting

The coefficients of the weighting matrix depend mainly on the geometrical locations of the measurement points for the inverse distance and the Kriging method. The weighting coefficients are almost independent of the current sensor data for these two methods.

Other modeling approaches set the weighing coefficients in order to give the best fit for a set of training data. A disadvantage of these approaches is that in addition to the values of the source points, the values of the destination points must also be known in advance for use as training data. Because the weighing coefficients depend on the sensor data of the training experiment, it is necessary to test by a cross-validation whether it is possible to apply the model to future experiments. Linear curve fitting is introduced as an example for this approach.

Equation (3) was transferred to a matrix form. The rows of the matrices $\hat{\mathbf{Z}}$ and $\mathbf{S}$ contain all measurement values at one sampling instance. The matrix $\mathbf{W}$ contains the time-invariant weighting coefficients:

$$\hat{\mathbf{Z}} = \mathbf{S} \cdot \mathbf{W} \tag{10}$$

The weighting coefficients are calculated as the least square error solution of the overdetermined linear set of equations (10) by the Moore-Penrose pseudoinverse [12]:

$$\mathbf{W} = \left(\mathbf{S} \cdot \mathbf{S}^{\mathbf{T}}\right)^{-1} \cdot \mathbf{S}^{\mathbf{T}} \cdot \mathbf{Z} \tag{11}$$

If the weighting matrix is recalculated by equation (11) for each experiment, the curve fitting delivers a superb approximation that is between 75% and 87% better than those calculated with Kriging (Table 2). However, the curve fitting fails to give a prediction for experiments in which the destination values are not known in advance. The weighting matrix was calculated with the data of experiment one and applied to experiment two and vice versa. The interpolation error of this cross-validation was always larger than the error predicted by Kriging. The prediction for the destination points in experiment one was hardly better than the simple average of source values.

Table 2. Interpolation error $\bar{\varepsilon}$ and cross validation for linear curve fitting

Experiment / Source points	Ex1 / 8	Ex2 / 8	Ex1 / 30	Ex2 / 30
Linear curve fitting	0.283 K	0.415 K	0.134 K	0.177 K
Cross validation curve fitting	2.638 K	2.234 K	2.417 K	1.470 K

This problem might also appear in other learning methods in which the parameters of the prediction system are trained by the data of one or more experiments. Their ability to predict destination values in future experiments has to be tested by careful cross-validation.

6 The Number of Required Sensors

The prediction error of the Kriging method decreases with the number of source points as shown in Table 1. In order to answer the initial question of this paper concerning the required number of sensors, the average prediction error of the Ordinary Kriging method was plotted in Figure 4 as a function of the number of source points. Starting with a configuration of one sensor in each of the eight corners, new sensors were added one by one. Furthermore, the effect of the location of the new sensors on

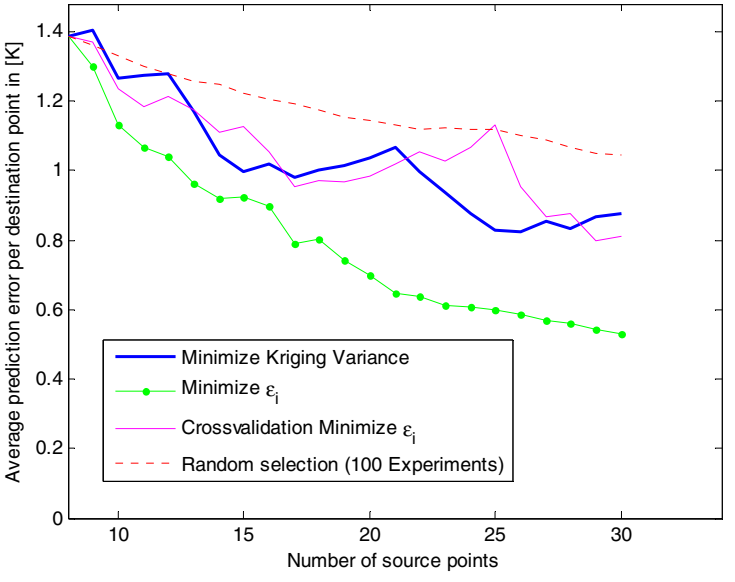


Fig. 4. Average prediction error $\bar{\varepsilon}$ as a function of the number of source points for different sensor-addition strategies

the prediction error was tested by this simulation. New points were added according to four different strategies:

a) New points were added randomly. The curve shows the average of 100 random experiments.

b) The Kriging method also provides the means to estimate the expected error in unknown destination points. The Kriging variance KV or its square root, the Kriging standard deviation σ_K, can be calculated as the product of the two vectors in equation (8):

$$KV = \sigma_K^2 = \begin{bmatrix} w_{1,q} \\ \vdots \\ w_{N_S,q} \\ \mu \end{bmatrix}^T \cdot \begin{bmatrix} v_{1,q} \\ \vdots \\ v_{N_S,q} \\ 1 \end{bmatrix} \tag{12}$$

The goal of this strategy b) is to minimize the average Kriging variance of the remaining destination points. All options to convert one destination point into a source point were tested. The point that resulted in the lowest average was selected as new source point.

c) The destination point with the highest deviation between the predicted and measured ε_i was converted into a source point.

d) Whereas the weighting coefficients are independent of the number of the experiment, the cold and hot spots as locations of temperature extremes can change location for different transports. Thus, the points with the

maximum ε_i are also subject to change. Therefore, strategy c) has to be tested by cross-validation. The sequence for adding new sensors was determined according to the measurements from experiment two by employing strategy c). Thereafter, the same sequence was applied to experiment one.

A share of the measured prediction errors ε_i results from calibration tolerances of the sensors at the destination points. The average measurement error was estimated by a test in a climatic chamber for a set of 36 TurboTag data loggers. Their temperature measurement exhibited a standard deviation of 0.25 K at 0 °C, 0.38 K at -10 °C, and 0.68 K at -25 °C. The average temperature in our experiments was approximately -25 °C. Therefore, the related tolerance of 0.68 K should be considered as the low boundary for the prediction error.

In general, not only the temperatures at the remaining destination points of the data set should be estimated, but also those at any point inside the cargo hold. Therefore, the simulation was stopped after 30 source points were added. Otherwise, there would not have been a sufficient number of destination points remaining to reliably calculate the average prediction error.

Only strategy c) goes slightly below the low boundary, because it selects the points directly according to their prediction error, thus disregarding sensors with high tolerances. The other three strategies showed a slower decrease of the prediction error. If new points are added by one of these strategies, far more than 30 source points are necessary to reach the low boundary.

The four strategies were compared on the basis of the average error $\varepsilon_{20\text{-}30}$ over the interval between 20 and 30 source points. Strategy c), in which new sensors are added according to the maximum ε_i, showed the best result with $\varepsilon_{20\text{-}30} = 0.60$ K. The cross-validation increased its error to $\varepsilon_{20\text{-}30} = 0.96$ K, which is slightly higher than the error that results from adding new sensors based on the Kriging variance with $\varepsilon_{20\text{-}30} = 0.91$ K. The improvement using the latter two strategies is less than 20% compared to the average of the random experiments with $\varepsilon_{20\text{-}30} = 1.11$ K.

In typical applications, little or no sensor data is available before installation. In these cases, it is only possible to determine the sensor positions based on the Kriging variance. Strategy c) can only be applied if a larger data set is available. The number of measurement points in the data set must exceed the number of the reduced sensor positions. The error of strategy c) might increase for data from untrained transports, but this error is only slightly larger compared to errors using other strategies.

Strategies a) to d) are all greedy in the sense that they seek only the instant advantage and cannot change or undo earlier decisions. A fifth algorithm was tested as an example of a less greedy variant of strategy b). The algorithm searches for the best combination of the following three steps in order to minimize the average Kriging variance for the remaining destination points: i) remove one source point; ii) replace it with another point; iii) add a further point as source point. But the advantage of this strategy compared to b) was rather marginal. For 30 points, both strategies resulted in the same list but ordered differently.

The number of sensors required to achieve a given limit for the average prediction error can be obtained from Figure 4. If, for example, the limit is 1.0 K and the positions are determined by the Kriging variance, at least 22 sensors are necessary.

7 Plausibility Testing Based on the Kriging Variance

The Kriging variance also provides a means of testing the plausibility of single measurements or of the interpolation process itself. This related type of cross-validation selects all measurements as source points except for one. The Kriging standard deviation $\sigma_K(i)$ and the prediction error ε_i are calculated for this destination point i. The process is repeated after selecting the next location as destination point. Figure 5 shows the resulting values for all destination points.

If the data set satisfies the statistical assumptions of the Kriging method, the average of σ_K should be equal to the average prediction error. A large difference between these two values indicates that either the expected value of the measurements exhibits a spatial dependency, that a spatial trend or an anisotropic relation of the variogram has not been considered, or that the variogram does not correctly represent the measurements.

The values of the Kriging standard deviation are slightly too high for our data set compared to the measured prediction error. This could be caused by the temperature trend that was observed in direction of the y-axis. If a trend exists which is not compensated in the Kriging process, the variogram can produce excessively high values according to Schafmeister [10, page 37].

The described cross-validation could also test whether the measured values of one or more sensors are plausible. Figure 5 shows four points in which the measured values differ by more than 2 K from the calculated prediction. These deviations can arise due to the following:

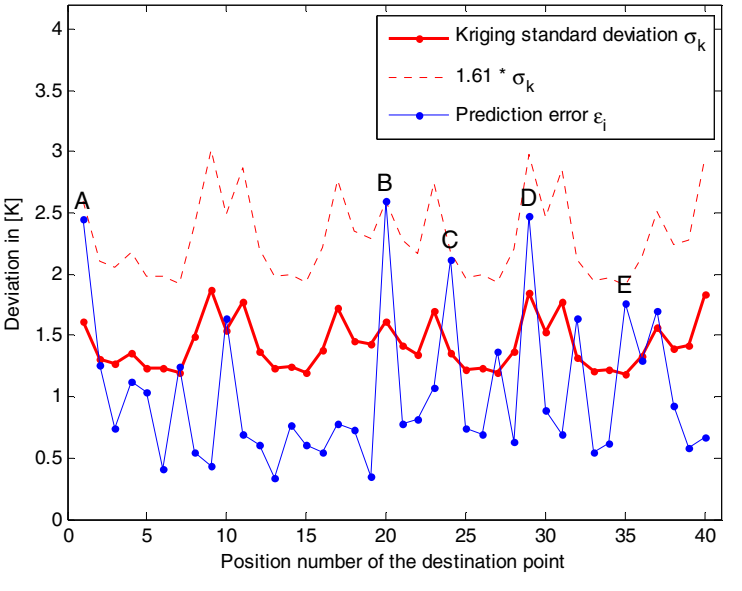


Fig. 5. Kriging standard deviation and prediction error as a function of the position of one destination point

a) The prediction itself has a high level of uncertainty, because the neighbor source points are too far away. This is expressed by the value of $\sigma_K(i)$.
b) The error is caused by random noise or statistical effects.
c) The prediction error has a physical cause. The point could be warmed up by a heat source that produces only a localized effect.
d) The sensor is faulty or exhibits an excessive tolerance.

In order to eliminate option a), the relation between the prediction error and the Kriging standard deviation was used as an indicator of the likelihood of a sensor fault at this point.

$$F(i) = \frac{\varepsilon_i}{\sigma_K(i)} \tag{13}$$

The five points with the highest indicator values $1.3 < F(i) < 1.61$ are marked with the letters A-E in figures 2 and 5. Point D has a higher prediction error ε_i than point E, although point E has a higher indicator value. The error at point E has to be considered as more significant. The distances to the two next neighbors are only half as great as the distances for point D. Therefore, the prediction at point E is expected to be more accurate.

Random noise might be the best explanation for the high indicator values of these points. If a Gaussian distribution is assumed, 81% of the values should be inside the interval $\pm 1.3 \cdot \sigma$. A share of 5 out of 40 sensors with F(i) > 1.3 can be explained by the statistical distribution, but other possible explanations for the high prediction errors should also be considered. Points A through D are all located close to the floor of the cargo hold. Presumably, the sensors are blocked by boxes in front of them. These sensors measure the box temperature rather than the distribution of air temperature.

The sensors that are most likely to be faulty can be identified by calculation the Kriging variance in order to check their plausibility. Sensors with a deviation much higher than the Kriging standard deviation should be carefully checked. In our data set, the deviations could be attributed to a physical cause or to noise effects. Except for these cases, sensors with high deviations should be regarded as faulty.

8 Summary and Conclusion

Wireless sensor networks can be used to detect local temperature deviations in cool chain transports, but oversampling of the cargo hold by implementing too many sensors should be avoided. The temperature value at any position can be estimated by interpolating the values of a limited number of sensors.

The Kriging method proved to be a useful tool for the evaluation of spatial sensor measurements. It delivers a more accurate interpolation than the commonly used inverse distance weighting. The expected prediction error at positions where no sensor is present can be calculated by using the Kriging variance. The minimum number of sensors can be estimated by a plot of the prediction error as a function of the number of measurement points. The plausibility checking based on the Kriging variance also provides a way to detect faulty sensors.

The disadvantage of Kriging is that it requires at least one data set with a high number of sensor positions to estimate the required variogram. The 40 positions of our data set seem to suffice for this process only by a small margin.

Although Kriging was originally developed for data sets made up of thousands of positions, the comparison of different interpolation methods showed that Kriging can also be usefully applied to typical sensor networks applications employing lower numbers of measurement points.

Acknowledgment

This research was supported by the German Research Foundation (DFG) as part of the Collaborative Research Centre 637 "Autonomous Cooperating Logistic Processes". We further thank Rungis Express AG, Germany for the provision of test facilities. See www.intelligentcontainer.com for additional project information.

References

1. Scheer, P.P.: Optimising supply chains using traceability systems. In: Smith, I., Furness, A. (eds.) Improving traceability in food processing and distribution, pp. 52–64. Woodhead publishing limited Cambridge, England (2006)
2. Koutsoumani, K., Taoukis, P.S., Nychas, G.J.E.: Development of a safety monitoring and assurance system for chilled food products. International Journal of Food microbiology 100(1-3), 253–260 (2005)
3. Tsironi, T., Gogou, E., Taoukis, P.: Chill chain management and shelf life optimization of MAP seabream fillets: A TTI based alternative to FIFO. In: Kreyenschmidt, J. (ed.) 3rd International Workshop on Coldchain Management, Bonn, pp. 83–89 (2008), http://www.ccm.uni-bonn.de
4. Moureh, J., Flick, D.: Airflow pattern and temperature distribution in a typical refrigerated truck configuration loaded with pallets. International Journal of Refrigeration 27(5), 464–474 (2004)
5. Tanner, D.J., Amos, N.D.: Heat and Mass Transfer - Temperature Variability during Shipment of Fresh Produce. Acta Horticulturae 599, 193–204 (2003)
6. Jedermann, R., Schouten, R., Sklorz, A., Lang, W., van Kooten, O.: Linking keeping quality models and sensor systems to an autonomous transport supervision system. In: Kreyenschmidt, J., Petersen, B. (eds.) Proceedings of the 2nd international Workshop on Cold Chain-Management, pp. 3–18. University Bonn, Bonn (2006)
7. Jedermann, R., Lang, W.: Semi-passive RFID and beyond: steps towards automated quality tracing in the food chain. International Journal of Radio Frequency Identification Technology and Applications (IJRFITA) 1(3), 247–259 (2007)
8. Krige, D.G.: A statistical approach to some mine valuations and allied problems at the Witwatersrand, Master's thesis of the University of Witwatersrand (1951)
9. Camilli, A., Cugnasca, C.E., Saraiva, A.M., Hirakawa, A.R., Corrêa, P.L.P.: From wireless sensors to field mapping: Anatomy of an application for precision agriculture. Computers and Electronics in Agriculture 58(1), 25–36 (2007)
10. Schafmeister, M.T.: Geostatistik für die hydrogeologische Praxis. Springer, Berlin (1999)
11. Chilès, J.P., Delfiner, P.: Geostatistics - modeling spatial uncertainty. John Wiley & Sons, New York (1999)
12. Ben-Israel, A., Greville, T.N.E.: Generalized Inverses. Springer, New York (2003)

Acoustic Sensor Network-Based Parking Lot Surveillance System

Keewook Na, Yungeun Kim, and Hojung Cha

Department of Computer Science, Yonsei University
Sinchon-dong 134, Seodaemun-gu, Seoul 120-749, Korea
{kwna,ygkim,hjcha}@cs.yonsei.ac.kr

Abstract. Camera-based surveillance systems are commonly installed in many parking lots as a countermeasure for parking lot accidents. Unfortunately these systems cannot effectively prevent accidents in advance but provide evidence of the accidents or crimes. This paper describes the design, implementation, and evaluation of an acoustic-sensor-network-based parking lot surveillance system. The system uses sensor nodes equipped with low-cost microphones to localize acoustic events such as car alarms or car crash sounds. Once the acoustic event is localized, the cameras are adjusted to the estimated location to monitor and record the current situation. We conducted extensive experiments in a parking lot to validate the performance of the system. The experimental results show that the system achieves reasonable accuracy and performance to localize the events in parking lots. The main contribution of our work is to have applied low-cost acoustic sensor network technologies to a real-life situation and solved many of the practical issues found in the design, development, and evaluation of the system.

Keywords: Acoustic source localization, Parking-lot surveillance system, Wireless sensor networks.

1 Introduction

Over the past few years, sensor-network-based parking lot management systems have drawn the attention of researchers. Lee et al. [1] proposed a parking lot application using magnetic and ultrasound sensors to accurately count the number of passing vehicles. Barton et al. [2] and Boda et al. [3] proposed parking lot management systems to find empty parking spaces. Although these systems enhanced the performance of conventional parking lot management systems by using various sensors, most neglected the security issues in parking lots. As the number of large parking lot increases, parking lot crimes such as car theft or vandalism also increase. However, most of the existing parking lot security systems have not efficiently protected these crimes. Parking lot security is generally maintained by car anti-theft systems or surveillance cameras that are pre-installed in a parking lot. These methods, however, have limitations in dealing with emergencies. In the case of camera-based surveillance systems, security guards should monitor the current situation in real time. Due to the practical restriction on constant monitoring of the system, camera-based

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 247–262, 2009.
© Springer-Verlag Berlin Heidelberg 2009

surveillance systems are normally used to provide evidence after accidents have occurred. SensEye [4] was proposed to address this issue. The system approximately detects abnormal movements using sensor nodes equipped with low-performance cameras. Once the movement is detected, a high-performance camera is activated to zoom in on the target. Since there are diverse types of moving objects in a parking lot, detecting abnormal movement is difficult. In the recently proposed SVATS (A Sensor network based Vehicle Anti-Theft System) [5], a small sensor network is constructed in a vehicle to identify unauthorized drivers. Although the system supports quick response and is resilient to attacks, the sensor nodes should be installed in advance. Moreover, the system cannot deal with other types of accidents except for vehicle burglars.

To address the aforementioned problems of the existing systems, we propose an acoustic sensor-network-based parking lot surveillance system. The system uses low-cost microphone-enabled sensor nodes to localize acoustic events such as car alarms or car crash sounds. Once the acoustic event is localized, the system makes use of cameras to pinpoint and monitor the event scene. The contributions of this paper are as follows:

- To the best of our knowledge, the proposed system is the first sensor-network-based parking lot surveillance system using the acoustic source localization algorithm.
- We proposed several mechanisms to improve the performance of the existing acoustic source localization algorithm for real parking lot environments.
- We improved the accuracy of the system by integrating image processing techniques, such as template matching and motion detection, into our sensor network-based system.
- The overall cost of the proposed system is significantly lower than the previous solutions since our system uses a small number of low-cost sensor nodes and cameras.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the proposed system. Section 3 describes the acoustic source localization algorithm and several enhancement techniques. Section 4 presents the integration with surveillance cameras. We present a detailed experimental environment and results in Section 5. We conclude the paper in Section 6.

2 System Overview

Figure 1 illustrates the overall structure of the proposed surveillance system. The system consists of three sub-systems: the Acoustic Source Localization (ASL) system, the surveillance camera system, and the server system. The acoustic source localization system localizes emergency sounds such as anti-theft alarms or car crash sounds. The estimated position of the acoustic event is delivered to the server, which displays the estimated position on the map and sounds an alarm. The surveillance camera immediately adjusts the camera toward the estimated position to capture the event scene, and performs motion detection to find a more accurate position.

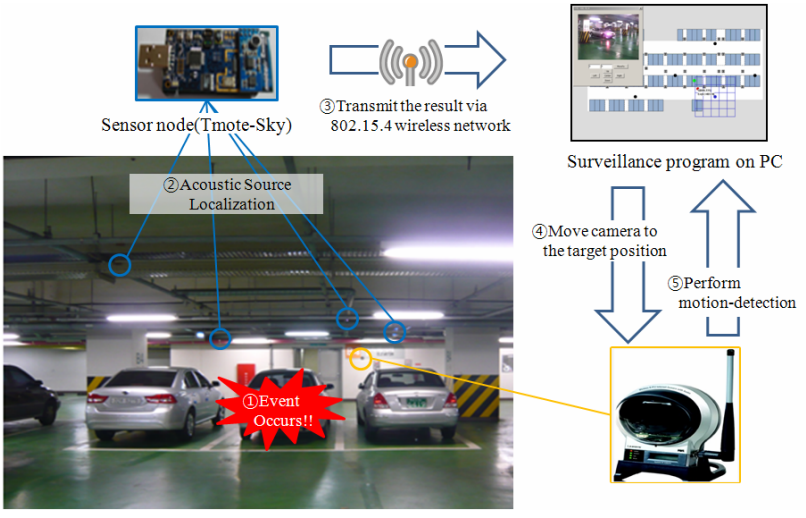


Fig. 1. System overview of the intelligent parking lot surveillance system

Microphone-enabled sensor nodes are attached to the ceiling of the parking lot and sample the current sound level at 3-kHz sampling rates. The Distributed Acoustic Source Localization (DSL) [6, 7] runs on each sensor node to localize the acoustic events. We present the detailed algorithm and enhancements in Section 3. In the surveillance camera system, a set of cameras is installed to cover the entire area of the parking lot. Considering diverse installation environments, we used the cameras that have both wired and wireless network interfaces. The surveillance camera system controls the camera direction to capture the estimated location of the acoustic source. The final location is accurately adjusted with the motion detection algorithm. The surveillance server notifies the supervisor of the event, through the video streamed from the camera system, which monitors the current situation. The server displays the estimated position of the event reported from the ASL system, and then sends the information to the surveillance camera system.

3 Acoustic Source Localization System

In this section, we describe the acoustic source localization algorithm used in our parking lot surveillance system. Several techniques to improve detection accuracy are also described.

3.1 Distributed Acoustic Source Localization

The key part in our system is to find the position of the acoustic source using low-cost sensor nodes. Among the previous techniques, we employed DSL (Distributed acoustic Source Localization), [6, 7] which is the distributed acoustic source localization algorithm. In DSL, sensor nodes exchange their sound detection times when the current sensing value is higher than the threshold. The position of an

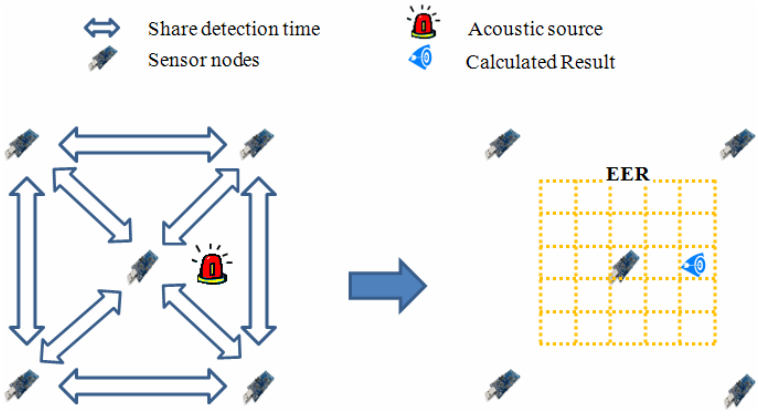


Fig. 2. The DSL algorithm

acoustic source is calculated using order of detection time among sensor nodes. DSL is able to localize sound events irrespective of sound types since the mechanism uses time information. The system requires no additional hardware components except for cheap microphone, in contrast with AOA (Angle Of Arrival) [8] or Beamforming[9].

Previous systems require expensive devices such as additional sampling circuit or high-performance microprocessor to obtain high sampling rates. For example, the shooter localization system [10] used a sensor board with Xilinx XC3S1000 FPGA chip, and the Acoustic ENSBox system [11] used an ARM-based CPU module.

Figure 2 shows the overview of the DSL algorithm [6] used in our parking lot surveillance system. When an acoustic event occurs, sensor nodes share their detection time to elect a leader node that has the earliest event detection time. The leader node calculates the EER (Essential Event Region) [7], which is the possible region of the source location. All nodes check which cell in the EER has a high probability of the acoustic source location by comparing detection times with each other and then transmit the result to the leader node. The leader node finally estimates the source location with the collected data. Detailed descripton on DSL and EER are found in our previous papers [6][7].

Initially, we used the original version of DSL. However, our preliminary experiments conducted at a parking lot did not show as a good performance as we had expected. The problem was revisited, and several error sources were found in the real experiments. The first error source is the low sampling rate of sensor node. The higher the frequency of acoustic events, the higher the errors in detection time. We therefore require an additional algorithm that complements the low sampling rate to accurately detect high-frequency acoustic sources. Second, the location of node deployment affects the localization errors. Since the sensor nodes located on the edge of the parking lot suffer from a lack of information needed to estimate the position of an acoustic source, DSL exhibits worse performance at the corner side than in the middle of the sensor field. The final error source is the duration of the acoustic event. The original DSL algorithm was designed to detect short-duration sound events; hence, one long-duration acoustic event is considered a series of short-duration acoustic events. Repeated detection of one acoustic event may cause energy waste,

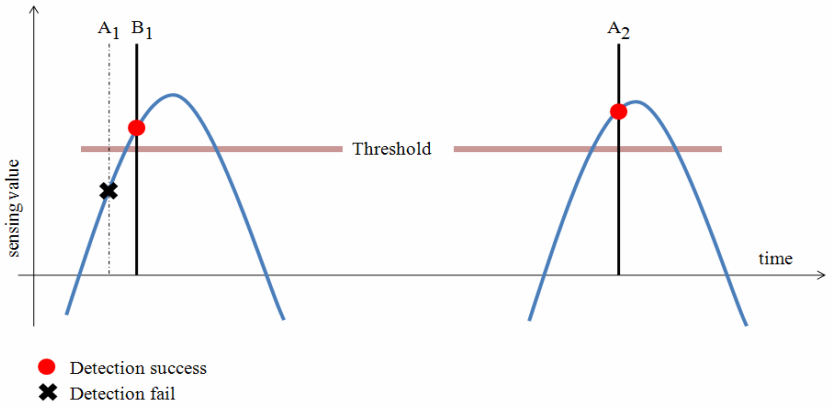


Fig. 3. Sampling time difference between two nodes

and different starting points of the next sensing cycle also cause significant localization errors.

3.2 Improving Accuracy on Detection Time

The low sampling rate of a microphone built into sensor nodes makes it difficult to accurately measure the sound occurrence time. The error rates increase especially in underground parking lots because of the strong echo and many obstacles. To compensate for the errors in detecting time, we propose a compensation algorithm by exploiting log data.

Figure 3 shows the wave form of an acoustic event. At time A_1, node A cannot detect an acoustic event since the sensing value is lower than the threshold. Therefore, node A will detect a sound event at time A_2. However, node B can detect a sound event at time B_1. This means that the phase of the sampling cycle and the threshold level affect the accuracy in detecting the correct time. To address this problem, our compensation algorithm determines the sound detection time using two levels of thresholds. The algorithm has three steps:

1) A buffer with size *n* keeps the measured sensing value in FIFO order.
2) When the current sensing value is higher than the default threshold T_1, the average of the sensing values in the first half of the queue is calculated. The calibrated threshold T_2 is then computed using Equation (1).

$$T2 = avg + \alpha < T1 \tag{1}$$

3) By traversing the second half of the buffer in the time order, the time upon which a sensing value higher than T2 was recorded is determined as the sound detection time.

As shown in Figure 4, the detection time is initially where the sensing value is 2080 with threshold T1. It is then corrected to the time when the sensing value is 1920 with

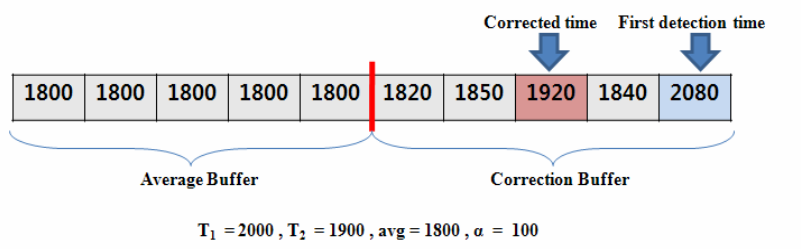


Fig. 4. Correction buffer

threshold T2. Finding the appropriate α value is an important issue. With a low α value, the algorithm is susceptible to surrounding noise. On the other hand, a high α value can cause a problem similar to that of the single threshold T1. In our system, we use the empirically-found α value of 100. With the proposed algorithm, we reduce the error between the detection time and the actual event time since the impact of surrounding noise is alleviated through two levels of thresholds.

3.3 Handling Lengthy Event Source

Repeated detection of one long-duration acoustic event causes energy waste by the sensor nodes. Moreover, different starting points of the next sensing cycle can be an error source. In Figure 5, for example, node 1 and node 2 detected an acoustic event at a similar time. However, different starting points of the next sensing cycle cause jitters, which can be a significant source of estimation errors.

To solve this problem, the proposed system stops sensor nodes from sensing the subsequent acoustic events. Only the node delivering the estimated position to the server continues the sampling to detect the end of the acoustic event. If there is no acoustic event that exceeds the threshold for the predefined time, the acoustic event is assumed to be finished. A message is then transmitted to activate the sampling of other nodes.

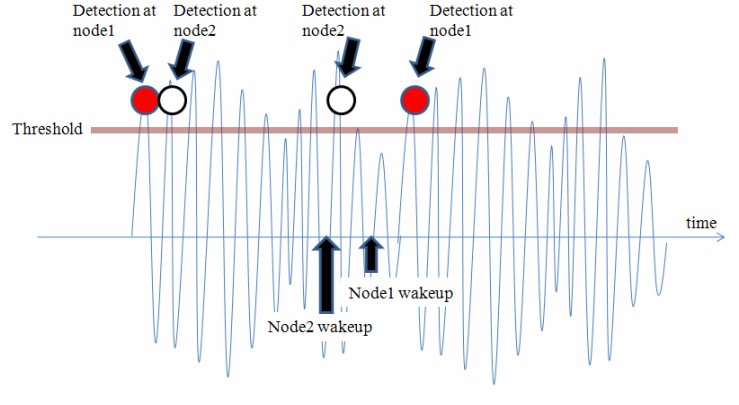


Fig. 5. Detection time error vs. wakeup time

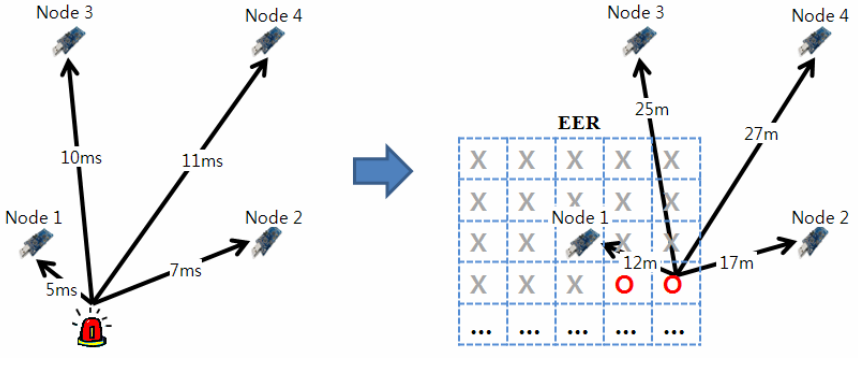
3.4 Improving Detection Accuracy on the Edge

In DSL, the node that has the earliest event detection time is elected as the leader node, and collects event detection times from neighboring nodes. Since the EER is decided based on the gathered information, the distribution direction of the neighboring nodes is important to estimate the position of an acoustic source. The performance of DSL varies depending on the position of an acoustic source. If an acoustic event occurs in the middle of the sensor field, the position is detected with low error. If an acoustic event occurs close to the edge, however, localizing an acoustic source is difficult because of the lack of information required to estimate the position.

To overcome the lack of information on the edge, the proposed algorithm considers the order of the event detection times as well as time differences between nodes. The original DSL uses only time information to determine which node detects the event earlier, whereas our algorithm exploits the differences in event detection times. The algorithm works as follows:

1) Calculate the EER of the leader node using the DSL algorithm and sort the detection times of all the nodes.
2) Divide the EER into discrete units of one parking space.
3) Perform the following procedures at each sub-region.
   (a) Calculate the distance from the current sub-region to each node.
   (b) Sort the distances.
   (c) If the order of the distances does not comply with the order of the event detection times, remove the current sub-region from the EER.
   (d) Finally, remove the current sub-region from the EER if the time difference and the distance difference sequences do not match.
4) Determine the position of an acoustic source by averaging the remaining sub-regions in the EER.

For example, in Figure 6(a), the event is first detected by node 1, followed by nodes 2, 3, and 4. The time differences are 2 ms, 3 ms, and 1 ms, respectively. As shown in Figure 6(b), the current sub-region in the EER satisfies the step 3 condition of the



(a) Acoustic source detection (b) Compare distance with detection time of each cell

Fig. 6. Algorithm on the edge

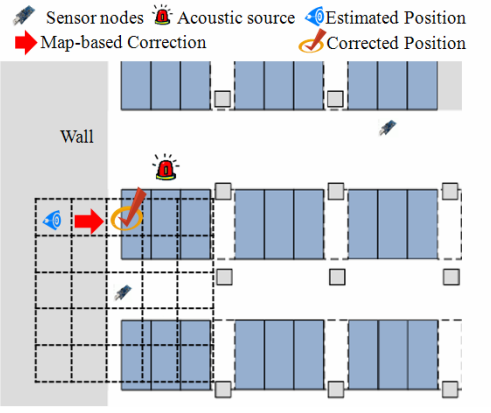


Fig. 7. Map-based localization result

algorithm because the order of the distances coincides with the order of the detection times, and the time and distance difference sequences match.

We conducted experiments to confirm the performance enhancement at the corner side. The average estimation error of our algorithm was 4.3 m, which is 48% of the original DSL. The proposed algorithm reduced approximately 30% of the communication overhead because an additional information exchange is not required after the leader node is elected. In other words, the proposed algorithm reduced the error rates at the corner side and the communication overhead.

3.5 Map-Based Correction

To reduce the location error of a sound event around obstacles, the proposed system exploits map information. The server maintains map information and performs a map-based correction when the estimated position of an acoustic source is received from the sensor nodes. If the estimated position is in the wall or fixed obstacles, the estimated position is discarded or moved. Figure 7 illustrates the correction method. Since the estimated position is in the wall, the method moves the position into the nearest obstacle-free region.

4 Surveillance Camera System

Recognizing the correct situation of a parking lot is difficult in practice if the system depends solely on warnings from the acoustic source localization system. The proposed system uses surveillance cameras to observe the scene and to increase the event detection accuracy. The camera transmits the actual view of the scene in real-time and tracks the scene by communicating with the server. For our system, a camera with wired and wireless communication capability is used for implementation.

Although various correction techniques are used in Section 3 for acoustic source localization, the result with the low-cost microphone sensors still exhibits errors. For

instance, in the case of a car collision, the position of an acoustic source and the position of the car may be different because the car can be moved by the force of the impact. The surveillance camera system should therefore cover the scene by considering the error range of the acoustic source localization. If the error range is unreasonably wide, the camera would transmit a picture of a very wide area. In our work, basic image processing techniques are supplemented to find an accurate position.

The first technique is to use the flickering headlights of a car. Typically, when the car alarm is activated, the headlights also flicker. A more accurate position can be found by analyzing the image taken by the camera and by checking if flickering headlights exist. To find the flickering headlights, we used OpenCV [12] library and template matching as in blink detection [13]. The process of finding the position of the flickering headlights is shown in Figure 8. The difference between the current picture and the previous picture from the camera is calculated and then compared with the template image that had been previously acquired. As a result, we determine the position of the headlights with the correlation maxima. According to our experiments, 68% of the maximum correlation value accurately indicates the position of the headlights.

The second technique is to use the motion detection feature that is provided by the surveillance camera. In general, when acoustic source is detected, there is a high possibility of movement being detected around the acoustic source. However, if the motion detection functionality is always active, any kind of normal movement is detected, and consequently, the power consumption increases. In our system, the motion detection feature is activated after the acoustic source is localized and the camera is adjusted to the spot. When the server system detects the acoustic source, the camera spots the detected area and performs motion detection.

The first technique detects the flickering headlights only, whereas the second one detects both flickering headlights and movements of objects. The second technique is used most of the time. However, the first one is exploited if a very accurate position of the car is required.

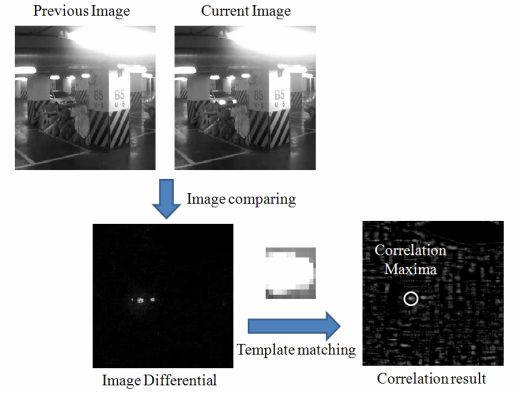


Fig. 8. Detect headlights via template matching

(a) Parking lot (b) Deployed sensor node

Fig. 9. Experiment environment

5 Evaluation

We discuss several experiments that were conducted to evaluate the system. Every experiment was performed in an underground parking lot environment. First, we performed preliminary experiments to find appropriate node spacing for the system deployment. The effect of radio congestion was evaluated, and the localization accuracy of acoustic event was also experimentally obtained. Finally, we conducted experiments to evaluate the overall system with every component activated in a multi-hop environment. In this experiment, the accuracy as well as the reaction time of the system were evaluated.

5.1 Experiment Setup

To evaluate the proposed system in a real environment, we deployed the system in the underground parking lot at our university, as shown in Figure 9 (a). A PC was used for the surveillance server, and we set up the surveillance cameras on the wall. The acoustic source localization system consisted of Tmote Sky [14] with a microphone sensor[15]. RETOS [16] was used for the operating system for the sensor nodes. FTSP[17] and ETA[18] are used for time synchronization in our work. We used ETA as our main time synchronization method since its implementation overhead is smaller than that of FTSP. The sampling rate for the sensor was 3 kHz, and the nodes were installed toward the ground so they could detect sound more accurately, as shown in Figure 9 (b).

5.2 Preliminary Experiments

First of all, to consider the node deployment issue, we performed an experiment that measured the system accuracy according to node spacing. Five nodes were deployed with node spacing varied from 8 m to 25 m. The result is shown in Figure 10. The average error was the biggest when the deployment range was 25 m. This is because the distance between two terminal nodes was 59 m so they could not communicate and obtain all the necessary information.

When the deployment range was shorter than 20 m, the radio communication problem did not occur, and the deployment range of 8 m had the best result. However, as the node spacing becomes narrow, the number of required nodes in the same space increases. In the case of 8-m node spacing, no fewer than 29 nodes are needed to

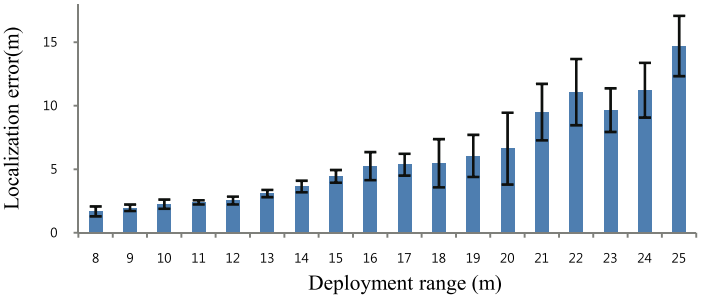


Fig. 10. Localization error vs. node deployment

cover a parking lot size of 88 m by 88 m. Considering both the cost and the accuracy, we decided to deploy 11 nodes in our parking lot size of 88 m by 50 m with spacing of 18.75 m, as shown in Figure 11. All the parking spaces were numbered from 1 to 35 to estimate the accuracy and response time at each parking space.

Second, we conducted experiments to decide the threshold value for the proposed system by measuring microphone levels for various sound events in a parking lot. The measured values from microphone are the ADC(Analog Digital Converter) readings. As Table 1 shows, the values of human dialogue and car moving sounds vary between 1700 and 2300. Consequently we set the threshold with 1600 to 2400 to filter these sounds. In case of slamming car door upon which the sensor node responds, the event can manually be classified with the camera. We plan to classify these sounds automatically using a high performance microphone attached to the camera.

Table 1. Sensing values of various sounds in a parking-lot

	Dialogue	Car moving	Slamming door	Car alarm
Decibel(dB)	65-70	75-85	90-110	105-125
Data from ADC	1780-2050	1715-2291	536-3502	75-3827

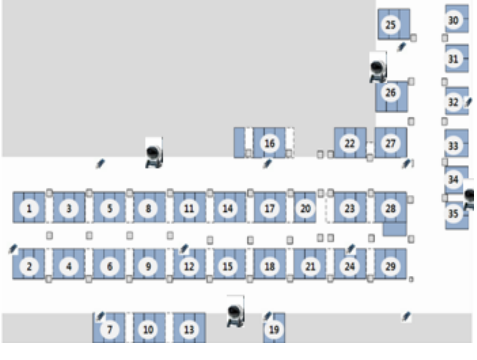
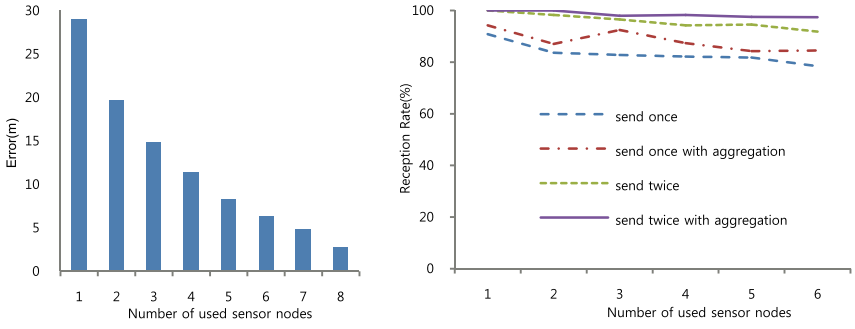


Fig. 11. Map of the parking lot



(a) Localization error (b) Reception rate

Fig. 12. DSL and radio congestion

5.3 Radio Congestion

In the proposed system, sensor nodes broadcast their detection time to the neighbor nodes when they detect an acoustic source. This simultaneous broadcasting can lead to radio congestion, and some nodes may suffer from a lack of the information needed to localize the acoustic source. This causes significant localization errors because the performance of the DSL algorithm strongly depends on the number of nodes that detect the acoustic source. Figure 12(a) shows the localization errors according to the number of nodes in the parking lot. The localization error significantly increases even with two or three nodes dropped out.

Our initial solution was to reduce congestion by having each node broadcast after a certain amount of delay, but this method becomes ineffective as the number of motes increases. The second solution was to use the data aggregation technique. Each mote broadcasts its information two times: first with only its own information, and second with information of other motes together with its own information. As shown in Figure 12(b), sending twice without aggregation is better than once, but the reception rate was still lower than 95% in the case of six nodes. Sending twice with aggregation, however, significantly reduced the congestion, and the reception rate was higher than 95% regardless of the number of nodes.

5.4 Acoustic Source Localization Accuracy

We measured the accuracy of acoustic source localization using 11 sensor nodes in a parking lot size of 88 m by 50 m. The experiment was performed 10 times in each region using a vehicle equipped with an alarm device. Figure 13 shows the localization errors in each parking space. The average error shown in Figure 13 is approximately 4 meters, which is an acceptable result considering the deployment spacing. The error for the outer regions such as Regions 1 and 2 is also as small as that for the central region so we identify that the correction algorithm is working appropriately. In Regions 7 and 35, however, the error was significantly increased because of the wall or pillars near the node.

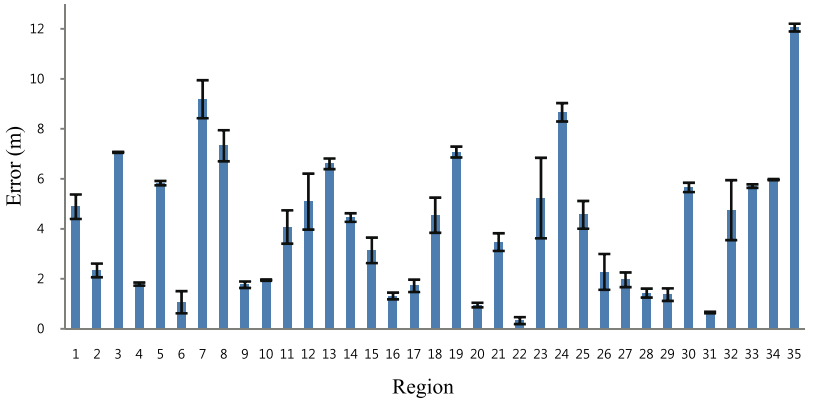


Fig. 13. Acoustic source localization accuracy

5.5 Overall System Accuracy

When an alarm is activated, the position is located using the camera. When the vehicle whose alarm is activated is in the camera picture, we regard it as "successful." Even though the calculated result from the sensor nodes accurately indicates the region in which the alarm occurs, the camera often cannot show the region due to various obstacles or the limitation of the camera angle. To remedy this situation, the cameras were installed symmetrically, and only two cameras that were the closest to the region that the alarm occurs are used to show the result.

With the laptop server, we can see the results moving around various regions. We deployed three router nodes in the experimental environment because the leftmost and rightmost nodes could not directly communicate with each other. Using multi-hop communications, we obtained the result in any region in the experimental environment. The detection result of the region with the alarm using a camera is shown in Figure 14. The car picture in the figure is the actual position of the car, and

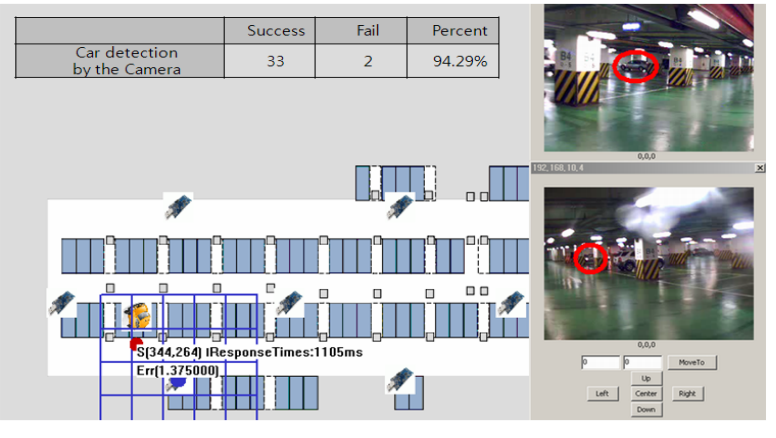


Fig. 14. Car alarm detection

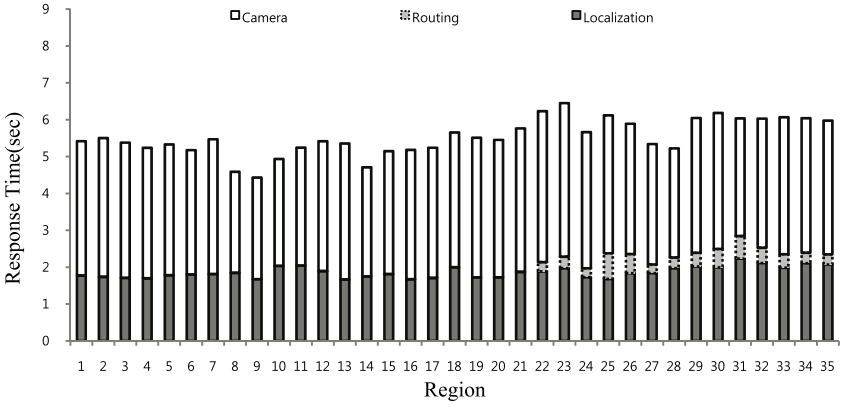


Fig. 15. Response time

the red dot is the estimated position. The success ratio was approximately 94%, which is considered fairly accurate in real environments.

5.6 Response Time

Experiments on reaction time upon events were conducted, and the result is shown in Figure 15. Since the proposed system exchanges time information with the surrounding nodes and the first node that sensed the sound transmits the result, the system imposes a delay in producing the actual result. The reaction time is, therefore, an important metric for the system performance. Figure 15 shows that the output time minimum was 1.67 seconds, the maximum was 2.23 seconds, and the average was 1.8 seconds, excluding the camera reaction time. The response times are influenced by radio congestion. As we discussed in Section 5.3, all nodes send data twice with time slot. With enhanced techniques for handling radio congestion, the response time would be reduced. The server waited for additional results for approximately one second after the reception of the result, because the result can be sent from a node that misunderstood itself as a first node. The camera movement includes motor actuation, which requires 1 to 3 seconds depending on the position of the camera. The overall reaction time including the camera movement was approximately 5 seconds. This mechanical delay is relatively long, but in practical situations this amount of delay is still much faster than a human being manually searches the area upon hearing an event. Additionally, the camera reaction time can be reduced by employing a high-performance camera which has a faster motor speed.

5.6 Discussion

The accuracy of the DSL algorithm strongly depends on the deployment topology of the network. We have in fact deployed the system in various topologies, but the accuracy error was not changed significantly. The reason is that we deployed the sensor nodes with more than 15-m node spacing in the underground parking lot where

many reverberation and obstacles exist; hence, the efficiency gain we obtained from node deployment was limited.

As shown in Figure 13, more than 8 m of error occasionally happened in the outer region regardless of the correction algorithm. This is due to the low sampling rate of 3 kHz and the surrounding obstacles, which hinder the accuracy of the detection time. Although this issue can be solved by using powerful sensor nodes with high-frequency sampling, it costs more, and therefore a trade-off between accuracy and cost exists in real system design and deployment.

In our work we did not conduct experiments on power consumption of the proposed system since the system is assumed to be deployed in parking lots where the external power supply is available.

6 Conclusion

In this paper, we proposed a parking lot surveillance system using microphone-enabled sensor nodes. The distributed acoustic source localization algorithm running on sensor nodes localizes the acoustic event in a parking lot. Once the acoustic event is localized, the surveillance camera system moves the camera direction toward the event and performs motion detection to correct the estimated position. The proposed system can localize loud sound events with a small number of sensor nodes and provide the event scene through video streaming in real time. We improved the existing acoustic source localization algorithm with several enhancement techniques, and achieved high accuracy in an underground parking that contained many pillars. The system effectively supervises a large parking lot of 100 vehicles using only a dozen sensor nodes. The experiments conducted in a real parking lot validated the feasibility of the system. Our future work includes further enhancement of the detection accuracy of events, as well as the classification of various acoustic sources.

Acknowledgments

This research was supported by the National Research Laboratory (NRL) program of the Korean Science and Engineering Foundation (No. M10500000059-6J0000-05910) and the ITRC support program supervised by the IITA (IITA-2008-C1090-0801-0015).

References

1. Lee, S., Yoon, D., Ghosh, A.: Intelligent Parking Lot Application Using Wireless Sensor Networks. In: International Symposium on Collaborative Technologies and Systems (2008)
2. Barton, J., Buckley, J., O'Flynn, B., O'Mathuna, S.C., Benson, J.P., O'Donovan, T., Roedig, U., Sreenan, C.: The D-Systems Project - Wireless Sensor Networks for Car-Park Management. In: Vehicular Technology Conference (2007)
3. Boda, V.K., Nasipuri, A., Howitt, I.: Design Considerations for a Wireless Sensor Network for Locating Parking Spaces: SoutheastCon (2007)

4. Kulkarni, P., Ganesan, D., Shenoy, P., Lu, Q.: SensEye: A Multitier Camera Sensor Network. ACM Multimedia (2005)
5. Song, H., Zhu, S., Cao, G.: SVATS: A Sensor-network-based Vehicle Anti-Theft System. In: Infocom (2008)
6. You, Y., Cha, H.: Scalable and Low-Cost Acoustic Source Localization for Wireless Sensor Networks. UIC (2006)
7. You, Y., Yoo, J., Cha, H.: Event Region for Effective Distributed Acoustic Source Localization in Wireless Sensor Networks. In: WCNC (2007)
8. Peng, R., Sichitiu, M.L.: Angle of Arrival Localization for Wireless Sensor Networks. In: Secon (2006)
9. Veen, B.D.V., Buckley, K.M.: Beamforming: A Versatile Approach to Spatial Filtering. IEEE ASSP Magazine (1988)
10. Volgyesi, P., Balogh, G., Nadas, A., Nash, C., Ledeczi, A.: Shooter localization and weapon classification with soldier-wearable networked sensors. In: Mobisys (2007)
11. Trifa, V.M., Girod, L., Collier, T., Blumstein, D.T., Taylor, C.E.: Automated wildlife monitoring using self-configuring sensor networks deployed in natural habitats. In: AROB (2007)
12. OpenCV Project, `http://sourceforge.net/projects/opencvlibrary/`
13. Chau, M., Betke, M.: Real Time Eye Tracking and Blink Detection with USB Cameras, Boston University Computer Science Technical Report No. 2005-12
14. Tmote Sky, `http://www.moteiv.com`
15. `http://www.panasonic.com/industrial/components/pdf/em05_wm62_a_c_cc_k_b_dne.pdf`
16. Cha, H., Choi, S., Jung, I., Kim, H.: RETOS: Resilient, Expandable, and Threaded Operating System for Wireless Sensor Networks. In: IPSN (2007), `http://retos.yonsei.ac.kr/`
17. Maróti, M., Kusy, B., Simon, G., Lédeczi, Á.: The Flooding Time Synchronization Protocol. In: Sensys (2004)
18. Kusy, B., et al.: Elapsed Time on Arrival: A simple, versatile, and scalable primitive for canonical time synchronization services. Int. J. of Ad Hoc and Ubiquitous Computing (2005)

Cooperative Intrusion Detection in Wireless Sensor Networks

Ioannis Krontiris[1], Zinaida Benenson[2,⋆], Thanassis Giannetsos[1], Felix C. Freiling[2], and Tassos Dimitriou[1]

[1] Athens Information Technology, 19.5 Km Markopoulo Avenue, Peania, Greece
{ikro,tdim,agia}@ait.edu.gr
[2] Laboratory for Dependable Distributed Systems, University of Mannheim, 68131 Mannheim, Germany
zina@uni-mannheim.de, freiling@informatik.uni-mannheim.de

Abstract. We consider the problem of cooperative intrusion detection in wireless sensor networks where the nodes are equipped with local detector modules and have to identify the intruder in a distributed fashion. The detector modules issue suspicions about an intrusion in the sensor's neighborhood. We formally define the problem of intrusion detection and identify necessary and sufficient conditions for its solvability. Based on these conditions we develop a generic algorithm for intrusion detection and present simulations and experiments which show the effectiveness of our approach.

Keywords: sensor networks, security, intrusion detection.

1 Introduction

The pervasive interconnection of autonomous and possibly wireless sensor devices has given birth to a broad class of exciting new applications in several areas of our lives, including environment and habitat monitoring, healthcare applications, home automation, and traffic control. At the same time, however, their unattended nature and the limited resources of their sensor nodes have created an equal number of threats posed by attackers in order to gain access to the network and the information transferred within.

There are several classical security methodologies so far that focus on trying to *prevent* these intrusions. A lot of work in sensor network security has focused on particular types of attacks and how they can be prevented. This can, however, only be a first line of defense. It is impossible, or even infeasible, to guarantee perfect prevention. Not all types of attacks are known and new ones appear constantly. As a result, attackers can always find security holes to exploit. For certain environments it makes sense to establish a second line of defense: An Intrusion Detection System (IDS) that can *detect* an attack and *warn* the sensors and the operator about it.

⋆ Zinaida Benenson was supported by Landesstiftung Baden Württemberg as part of Project "ZeuS" and by the Schlieben-Lange scholarship of the European Social Fund and the Bundesland Baden-Württemberg.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 263–278, 2009.
© Springer-Verlag Berlin Heidelberg 2009

1.1 Related Work

Intrusion detection has received some attention in wireless sensor networks before. Most work has focused on *local detection*, i.e., allowing nodes to locally detect specific attacks which are performed in their neighborhood [1,2,3,4,8,5,10].

Da Silva et al. [2] and Onat and Miri [10] propose similar IDS systems, where certain monitor nodes in the network are responsible for monitoring their neighbors. They listen to messages in their radio range and store in a buffer specific message fields that might be useful to an IDS system running within a sensor node. Kargl et al. [3] focus on the detection of *selfish* nodes that try to preserve their resources at the expense of others. Loo et al. [8] and Bhuse and Gupta [1] describe two more IDSs for sensor networks. Both papers assume that routing protocols for ad hoc networks can also be applied to WSNs.

In all the above work, there is no collaboration among the sensor nodes. The only collaborative approaches we are aware of focus on the local detection of selective forwarding attacks [4] and sinkhole attacks [5].

More extensive work has been done in intrusion detection for ad hoc networks [9]. In such networks, distributed and cooperative IDS architectures are also preferable. Detailed distributed designs, actual detection techniques and their performance have been studied in more depth. While also being ad hoc networks, wireless sensor networks are much more resource constrained. We are unaware of any work that has investigated the issue of intrusion detection in a general collaborative way for wireless sensor networks.

1.2 Contributions

In this paper we study a general approach to intrusion detection in wireless sensor networks which focuses more on the collaboration of sensors than on the detection of specific attacks. We believe that wireless sensor networks with their inherent redundancy are ideal for applying cooperative techniques. We therefore abstract from concrete local detection techniques and define in Section 2 the notion of a local *alert module*. Such modules issue suspicions about an intrusion in the sensor's neighborhood. We focus here on the case where sensor nodes try to detect a *single* malicious node, as this case is already surprisingly complex.

We formally define the problem of intrusion detection in Section 3 and identify necessary and sufficient conditions on the behavior of the local alert modules such that they contain enough information to cooperatively solve the intrusion detection problem in Section 4. These conditions also identify scenarios in which cooperative intrusion detection is unsolvable.

We further develop an algorithm that solves intrusion detection based only on the output of the alert modules in Section 5. The idea is that nodes in the neighborhood of the attacker exchange information about who they suspect and jointly identify the attacker. Note that this is not an easy task since the attacker can also participate in the protocol and try to bring honest nodes to a wrong conclusion. In this sense, intrusion detection may at first sight seem similar to the problem of Byzantine agreement [11]. However, we show that both problems are incomparable.

Finally in Section 6 we investigate the probability that symmetry conditions that make intrusion detection impossible to solve occur in practice, using simulations. In Section 7, we present a lightweight implementation of our cooperative intrusion detection algorithm on Moteiv Tmote Sky motes justifying the practicality of our approach.

2 System Model

2.1 Sensor Nodes and Communication

We present a strong system model used for proofs of necessary and sufficient conditions for intrusion detection in the sequel. It is useful, because if some problem is impossible to solve in our model, it is also impossible to solve in weaker models which are closer to the reality.

In our model, a wireless sensor network consists of a set $S = \{s_1, s_s, \ldots, s_n\}$ of n sensor nodes. Sensors communicate by sending messages over a wireless broadcast medium, meaning that if some sensor s sends a message, many other sensors can receive the message simultaneously. Possible collisions on the wireless medium can only delay the receipt of a message for a relatively short time.

For any sensor node s, the set of nodes with which it can directly communicate is denoted by $N(s)$. For simplicity, we assume a static and symmetric neighborhood relation, i.e., if $s \in N(s')$ then $s' \in N(s)$. We assume that every node knows its 2-hop-neighborhood.

Although the above assumptions are strong, considering unreliable asymmetric wireless links and frequent neighborhood changes in real sensor networks, we use them especially for proofs. In Section 4.4 we discuss how the system model can be weakened. We also present an implementation of our algorithms for real sensor networks in Section 7 which does not make these strong assumptions.

We make no assumptions about the communication topology defined by all neighborhood sets apart from that all nodes that behave according to the protocol (honest nodes) are connected via a path consisted only of other honest nodes. We expect that in typical sensor networks the density of the nodes is rather high so that this condition will be satisfied with high probability.

2.2 Attacker Model

We assume that an attacker can capture at most t nodes to launch an attack on the sensor network. We model this by allowing at most t nodes to behave in an arbitrary manner (Byzantine failure). However, we do not propose a Byzantine Agreement protocol, but focus on the Intrusion Detection Problem (Definition 3). The relationship between Intrusion Detection and Byzantine Agreement is shown in Section 4.3.

In the following, we concentrate on the case where $t = 1$. In this case, we call the captured node the *source* of the attack, or the attacker, and use the predicate $source(s)$ which is true if and only if (iff) s is the attacker. All other nodes are called honest: $honest(s) \equiv \neg source(s)$. As the rigorous examination of

this case already gives very useful insights on solvability of intrusion detection (and turns out to be quite complex), we leave the case $t > 1$ to future work.

2.3 Alert Module

Attacks are locally detected by a local intrusion detection mechanism. We abstract such mechanisms of a sensor node into an *alert module*. Whenever the alert module at node s notices something wrong in its neighborhood, the alert module outputs some set $D(s)$ of *suspected sensors*, called the *suspected set*. If $|D(s)| = 1$, then the node has identified the attacker. Most often however, $D(s)$ will contain a larger set of neighbors or may even be equal to $N(s)$.

For example, in the detection of the selective forwarding attack [4], the nodes are able to identify the one node that drops the messages by monitoring the transmissions in their neighborhood ($|D(s)| = 1$). On the other hand, in the protocol for detecting the sinkhole attack [5] nodes only know that the attacker has to be one of their neighbors ($D(s) = N(s)$).

Formally, the alert module satisfies the following properties:

- If the alert module at a honest node s outputs $D(s)$, then the source is in that set, i.e., $\exists s' \in D(s) : source(s')$.
- If the attacker attacks, then within some time delay δ the alert module at some sensor s outputs a set $D(s)$.
- Only neighbors are suspected by honest nodes, i.e., $\forall s \in S : honest(s) \Rightarrow D(s) \subseteq N(s)$.

If the alert module at some node s outputs some set, we call s an *alerted node*. The predicate A on S denotes the set of alerted nodes, i.e., $A(s)$ holds iff s is an alerted node. The set of alerted nodes is called the *alerted set*. Note that the attacker may or may not belong to the alerted set, depending on the strategy that the attacker chooses to follow. Also, we do not require that *all* neighbors of the attacker belong to the alerted set.

3 The Intrusion Detection Problem

The cooperative intrusion detection process is triggered by an attack and by the subsequent alerts by the local alert modules of the neighboring sensors. The process ends by having the participating sensors jointly *expose* the source.

More formally, the predicate $expose_s(s')$ is true if node s exposes node s'. The intrusion detection problem can now be defined as follows:

Definition 1 (Intrusion Detection Problem (IDP)). *Find an algorithm that satisfies the following properties:*

- *(Correctness) If an honest node s exposes a node s', then s is in the alerted set and s' is the source, i.e., $\forall s \in S : honest(s) \wedge expose_s(s') \Rightarrow A(s) \wedge source(s')$.*
- *(Termination) If the attacker attacks, then at most after some time τ all honest nodes in the alerted set expose some node.*

4 Conditions for Solving Intrusion Detection

The idea of cooperative intrusion detection is to exchange the outputs of local alert modules, thereby narrowing down the set of possible nodes that could be the attacker. In the following we assume that nodes have no other way to learn anything about the attacker than using their alert modules. As an initial example, consider the case depicted in Fig. 1(a). Node p suspects the source q, i.e., $D(p) = \{q\}$. Node q can claim to output $D(q) = \{p\}$. But p implicitly knows that it is honest, so it will ignore the information provided by q and expose q.

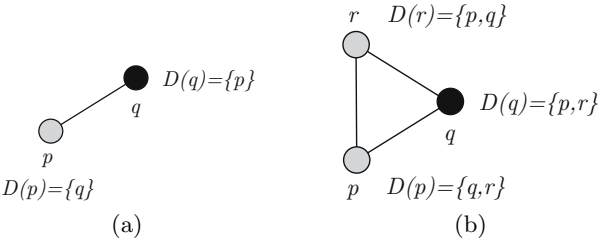


Fig. 1. Different types of alerted neighborhoods. Sources of attacks are marked black.

In the example in Fig. 1(b), however, three nodes p, q, and r all suspect each other (node q is the source). As every node occurs in exactly two suspect sets, p cannot distinguish node r from node q by using only the suspect sets. We conclude that it is impossible to solve IDP in this case.

Generalizing these two examples, the question arises about general conditions for the solvability of the intrusion detection problem. In the following, we give necessary (Section 4.2) and sufficient (Section 4.1) conditions for solvability of IDP using a deterministic algorithm for $t = 1$. We show the relationship between IDP and Byzantine Agreement in Section 4.3. In Section 4.4 we consider how weaker system models (unreliable links, neighborhood changes) affect the solvability of IDP.

4.1 Sufficient Conditions for Solving IDP

The Intrusion Detection Condition (IDC) We now give a sufficient condition for IDP solvability for $t = 1$ and deterministic algorithms. The intuition behind the condition is a generalization of the observation made in Fig. 1(b): If the suspected sets about some node s are structurally equivalent to those of the source, then the problem is in general not solvable.

Formally, for a node s we define the set $AN(s)$ to be the set of *alerted neighbors* of s, i.e.:

$$AN(s) = \{t | A(t) \wedge t \in N(s)\}$$

Furthermore, we define the set of alerted neighbors of p with respect to q $\tilde{AN}(p, q)$ to be the set of alerted neighbors of p without q, i.e.:

$$\tilde{AN}(p, q) = AN(p) \setminus \{q\}$$

For example, in Fig. 1(b) all three nodes are in alert mode and $AN(s) = D(s)$. The value of $\tilde{AN}(p,q) = \{r\}$ is the information content of $AN(p) = \{q, r\}$ that is valuable to q.

Definition 2 (Intrusion Detection Condition (IDC)). *The intrusion detection condition (IDC) is defined as:*

$$\forall p, q \in S : source(q) \Rightarrow \tilde{AN}(p,q) \neq \tilde{AN}(q,p)$$

Roughly speaking, IDC means that no other node has the same alerted neighborhood as the attacker. Note that if p and q are not neighbors, then IDC simplifies to:

$$\forall p, q \in S : source(q) \Rightarrow AN(p) \neq AN(q)$$

Theorem 1 (Sufficiency of IDC). *If $t = 1$, IDC is sufficient for ID, i.e., if IDC holds then IDP can be solved.*

Proof. Let all alerted nodes exchange their suspected sets. This is possible in our system model because each pair of honest nodes is connected by a path consisting of honest nodes, and communication is reliable.

Note that the attacker also can go into alert mode. Moreover, it can send different suspected sets to different nodes. However, as we assume that all nodes know their 2-hop-neighborhood, the suspected set of the attacker may only contain its neighbors. Otherwise, the attacker's messages would be discarded.

Consider the suspected sets received by an honest node p. If some node is suspected by a majority of the nodes, it is immediately identified as the attacker, because the attacker is included in the suspected set of every honest node.

A more complicated case arises when there are two or more nodes which are suspected by the majority of nodes. This situation can arise, e.g., if the attacker also goes into the alert mode and accuses some of its neighbors.

We denote the attacker as q. Assume that there is a node $p \neq q$ which is suspected by the same number of nodes as q. How can a node r distinguish between q and p?

(1) If $p = r$, then r knows that it is honest, and exposes q.

(2) Consider $p \neq r$. If all honest nodes suspect p, then the IDC does not hold. Thus, for some honest node s holds: $p \notin D(s)$ and $q \in D(s)$. It follows that q is alerted and $p \in D(q)$, as the number of nodes which suspect p is the same as the number of nodes which suspect q.

Node r must now decide which of nodes s and q lies about their suspicion. We now show that there is an alerted node v which is not neighbor of s. Indeed, if all alerted nodes were neighbors of s, then s and q would have the same alerted neighborhood with respect to each other, which contradicts the IDC. Thus, node r has to find out which of the nodes s and q is not a neighbor of some alerted node. This is possible as all nodes know their 2-hop neighborhood. This node has to be honest, and the remaining node is identified as the attacker. □

As an example, consider Figure 2. Nodes s and r are honest nodes and alerted. Node p is also honest, but not alerted. The attacker is node q, which is alerted.

In this example, nodes p and q are both suspected by two nodes. How can node r distinguish the attacker? IDC holds here, and node p is suspected by each of q and r. Thus, either q or s is lying about their suspicions. However, nodes r and s are not neighbors, and therefore, s cannot be the attacker.

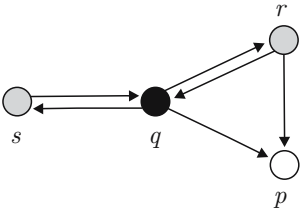


Fig. 2. Node q is the attacker, nodes s, r and q are alerted, while p is not alerted and it is marked white. $x \rightarrow y$ means that node x suspects node y. $D(r) = \{q, p\}$, $D(q) = \{p, r, s\}$, and $D(s) = \{q\}$.

The Neighborhood Conditions (NC). What happens if IDC is not satisfied? Can IDP still be solved, or is IDC also a necessary condition for solving IDP?

In the following we show that IDC is not a necessary condition. We give another sufficient condition for IDP solvability which can be valid in the network independently of the validity of the IDC.

Definition 3 (Neighborhood Conditions (NC)). *The Neighborhood Conditions (NC) consist of two conditions:*

- NC_1. *All neighbors of the attacker are alerted.*
- NC_2. *If two or more nodes are suspected by the majority of nodes, then all honest nodes suspected by the majority have non-alerted neighbors.*

Theorem 2 (Sufficiency of NC). *If the NC holds, then the IDP can be solved.*

Proof. We give an informal reasoning here. Let all alerted nodes exchange their suspected sets. If only one node is suspected by the majority of nodes, then this node is the attacker, as all neighbors of the attacker are alerted (NC_1). If there are two or more nodes which are suspected by the majority, the nodes in the alerted set have to find out which of these nodes have non-alerted neighbors. According to NC_2, only the attacker does not have non-alerted neighbors. □

4.2 Necessary and Sufficient Conditions for Solving IDP

We now show that for the solvability of IDP either the IDC or the NC (i.e., NC_1 and NC_2) should be satisfied in the sensor network.

Theorem 3. *IDP can be solved using a deterministic algorithm if and only if IDC or NC holds.*

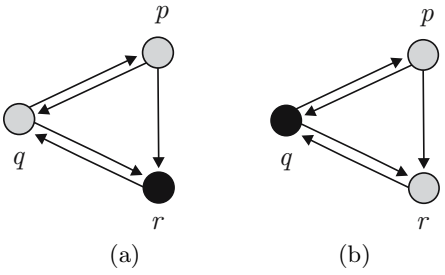


Fig. 3. Case (a): Node p suspects q and r, node q suspects p and r, node r is the attacker and suspects q. IDC and NC_2 are not satisfied. Case (b): The suspicions remain as in case (a), but node q is the attacker. No algorithm for solving the IDP can distinguish between (a) and (b). Therefore, it is impossible to expose the attacker.

Proof. As shown in Theorems 1 and 2, if IDC holds or if NC holds, then the intrusion detection problem can be solved (sufficiency). We now show that it is also necessary for the solvability of the IDP that the IDC *or* the NC holds. It suffices to show that if the IDC does not hold and the NC does not hold, then the IDP cannot be solved.

Assume that the above claim is not true. That is, there exists a deterministic algorithm $\mathcal{A}$ that always exposes the attacker in case both the IDC and the NC do not hold. Consider Figure 3(a). The IDC does not hold there because $\tilde{AN}(p,r) = \tilde{AN}(r,p) = \{q\}$. Also NC does not hold, because NC_2 does not hold: The attacker r and the honest node q are suspected by two nodes, but q does not have non-alerted neighbors. In this case, the algorithm $\mathcal{A}$ should expose r. However, the situation in Figure 3(b) is exactly the same as in (a) from $\mathcal{A}$'s point of view. The suspicions remain the same, the topology also does not change. Thus, there is no additional information to help $\mathcal{A}$ to distinguish between situations (a) and (b). However, $\mathcal{A}$ should be able to distinguish between (a) and (b) and to expose r or q accordingly. It follows that $\mathcal{A}$ does not exist. □

4.3 Byzantine Agreement vs. Intrusion Detection

In IDP, the honest nodes have to jointly expose the attacker. That is, they have to reach agreement on the attacker's identity. Although this looks similar to Byzantine Agreement [11], these two problems cannot be reduced to each other. In some cases, Byzantine Agreement can be solved whereas Intrusion Detection is not solvable, and vice versa.

Consider Figure 4(a). Here, node q is the attacker and suspects both p and r. The honest nodes, on the other hand, both suspect q. In this case, Intrusion Detection is trivially solvable. However, Byzantine Agreement for three participants with $t = 1$ cannot be solved [11]. In Figure 4(b) all nodes suspect each other. IDC does not hold for nodes s and q, NC also does not hold, as no node has non-alerted neighbors. Thus, Intrusion Detection is not solvable. However, Byzantine Agreement for $t = 1$ can be solved here [11].

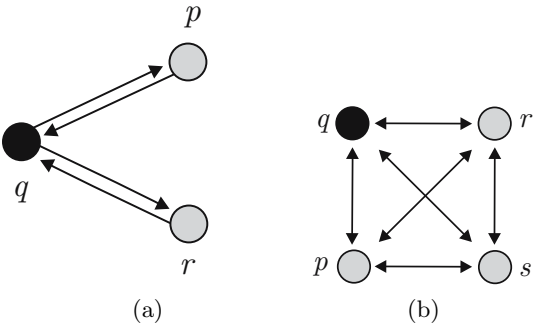


Fig. 4. Byzantine Agreement and Intrusion Detection cannot be reduced to each other. Case (a): Honest nodes p and r both suspect only the attacker q, thus Intrusion Detection can be solved, but Byzantine Agreement is not solvable. Case (b): Intrusion detection cannot be solved, Byzantine Agreement is solvable.

4.4 Solving IDP in a Weaker System Model

In proofs we used our assumptions on reliable and timely communication. In principle, we can also use weaker system models as long as they allow some protocols for reliable and timely exchange of suspected sets with high probability. For example, in our implementation we use an advertise-request protocol.

We also assumed a static and symmetric neighborhood relation. This assumption can also be weakened. All we need is that the nodes have secure information on their 2-hop neighborhood which does not change during a particular protocol run. In our implementation we let the nodes to find out their neighborhood in the secure initialization phase, where the attacker is absent. The neighborhood tables which are used for intrusion detection are then fixed. In case that a neighbor crashes, it looks in the protocol as if the node was not alerted. This can be tolerated as long as the IDC holds (the NC does not hold in this case). On the other hand, if some new neighbors arrive, they can be ignored. This is problematic, however, in case the new node is the attacker. A better solution would be to have a secure protocol for the neighborhood table update. We leave this to future work.

5 A Cooperative Intrusion Detection Algorithm

Based on the ideas of Section 4, we now develop a general algorithm to solve the intrusion detection problem, i.e., all honest and alerted neighbors of an attacker share their partial views, agree on the identity of the source and expose it.

5.1 Initialization Phase

Prior to the deployment, each node is preloaded with a one-way key chain of length l, using a pre-assigned unique secret key K_l. A one-way key chain [7]

$(K_0, K_1, \ldots, K_{l-1}, K_l)$ is an ordered list of cryptographic keys generated by successively applying a one-way hash function F (in our case SHA-1) to the key seed K_l, such as $K_j = F(K_{j+1})$, for $j = l - 1 \ldots 0$. Therefore, any key K_j is a commitment to all subsequent keys K_i, $i > j$; more specifically, K_0 is a key chain commitment for the entire chain. The length of the key chain l is a system parameter. In our implementation, we store the key chain on the external flash memory, such that it does not affect the memory requirements of our algorithm. This also allows us to set l to a large number, such that we can avoid the overhead of renewing the key chain during deployment.

The initialization phase takes place right after the network deployment. The duration of this phase is short enough so that we assume the absence of the attacker. We also require that all nodes discover their immediate neighbors, which is a standard procedure after deployment in almost all routing protocols. Further, all nodes discover their 2-hop neighborhood by broadcasting their IDs with a packet that has a TTL field equal to 2. The discovered neighborhood information is stored in the 2-hops neighborhood table. Then, each node announces their key chain commitment K_0 to all its 1-hop and 2-hop neighbors.

5.2 Voting Phase

During the voting phase each node in the alert region sends its vote to all the other members and respectively collects their votes. Let us denote the vote message from node s as $m_v(s)$. Each vote consists of the nodes suspected by the sender, so for node s, $m_v(s) = id||D(s)$. Node s "signs" its vote calculating the MAC with the next key K_j from its one-way key chain, and broadcasts

$$m_v(s), MAC_{K_j}(m_v(s)).$$

Following that, it sets a timer T_v to expire after time τ_v. During that time it waits to receive the votes of the rest of the alerted nodes and buffers them, as it has to wait for the key publishing phase in order to authenticate them.

The vote of each alerted node needs to reach all other alerted nodes. Since the messages are signed with a key known only to the sender, the attacker cannot change the votes. However, the attacker may refuse to forward votes, such that they must be forwarded through other paths, bypassing the attacker. Note that these paths can consist of more than two hops.

To ensure that the votes propagate to all alerted nodes, we follow a broadcast message-suppression protocol, similar to SPIN [6]. When an alerted node receives a vote, it advertises it, by broadcasting an ADV message. Upon receiving an ADV, each neighboring node checks to see whether it already has received or requested the advertised vote. If not, it sets a random timer T_{req} to expire, uniformly chosen from a predetermined interval. When the timer expires, the node sends a REQ message requesting the specific vote, unless it has overheard a similar REQ from another node. In the latter case, it cancels its own request, as it is redundant.

5.3 Publish Key Phase

In the Publish Key phase each node broadcasts the key of its hash chain, K_j, which was used to sign the vote. When a node receives the disclosed key, it can easily verify the correctness of the key by checking whether K_j generates the previous key through the application of F. If the key is correct, it replaces the old commitment K_{j-1} with the new one in its memory. The node now uses the key to verify the signature of the corresponding vote stored in its buffer from the previous phase. If this process is successful, it accepts the vote as authentic.

We allow sufficient time for the nodes to exchange their keys by setting a timer T_p. This timer is initialized just after a node publishes its own key and it is set to expire at time τ_p. During this time period, the nodes follow the same ADV-REQ scheme that we described above.

When the timer expires, the nodes move to the final step of processing the votes and exposing the attacker. In the case where a key has been missed, the corresponding vote is discarded.

Since nodes are not time synchronized, and some nodes may start publishing their keys while others are still in the voting phase, we need to consider "man in the middle" attacks. When a node sends its vote, an attacker may withhold it until that node publishes its key. Then it can change the vote, sign it again with the new key, and forward it to the next alerted node. Following that, the attacker also forwards the key, and the receiver will be able to verify the signature and accept the fake vote as authentic. We deal with this problem implicitly by relying on *residual paths* amongst the nodes. As votes are forwarded by all nodes, even if an attacker refuses to forward a vote, it will arrive to the intended recipients via other paths.

5.4 Exposing the Attacker

When each alerted node $s_1, s_2, \ldots, s_n$ has collected and authenticated the votes from all the other members of the alert region, it will have knowledge of all the corresponding suspect lists, $D(s_1), D(s_2), \ldots, D(s_n)$, its own included. Then it applies a count operator which counts the number of times δ_i each node i appears in the suspect lists, in order to produce the final intrusion detection result, i.e., the attacker's ID. All alerted nodes will reach the same result, since they all apply the same operator on the same sets.

As we proved in Section 4.1, IDC is a sufficient condition for the intrusion detection problem. But if it does not hold, then NC needs to hold in order to successfully identify the attacker, as we proved in Section 4.2. Thus, if there is one node holding the majority of the nodes, we know that this is the attacker. If not, then the honest nodes which also collected the majority have non-alerted neighbors. So, the nodes move to a new phase, the external ring reinforcement phase, where these neighbors are called to support their honest neighbors. We describe this phase in Section 5.5, where we will see that in this case the attacker is revealed.

5.5 External Ring Reinforcement Phase

As we said, when there are other nodes that have the same set of alerted neighbors $\tilde{AN}$ with respect to the attacker, i.e., IDC does not hold, the voting pro-

cess may be inconclusive, if these nodes collect the same number of votes. In this section, we present an algorithm where, if NC holds, the alerted region can distinguish amongst the prevailing candidates and find the actual one. So, for what follows we assume that NC holds, meaning that *all* neighbors of the attacker are alerted and that honest nodes collecting the majority of the votes have non-alerted neighbors.

Let us assume the set $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ of the nodes collecting the same number of votes as the attacker, including the attacker itself. According to NC, the nodes in $\mathcal{P}$ do not have exactly the same neighborhood. Honest ones will also have other neighbors, which are not in alerted state and are going to play an important role in this phase. Therefore, we make the following definition:

Definition 4 (External Ring). *The external ring is defined as the set of nodes which are not members of the alerted region, but any of them is a direct neighbor of at least one alerted node.*

Figure 5 shows an example where nodes 96 (the attacker) and 76 have the same alerted neighborhood, and therefore collected the same number of votes during the voting phase, i.e., $\mathcal{P} = \{96, 76\}$. The circle in the figure shows the neighborhood of the attacker. The nodes in the external ring are represented by a triangle. The neighborhood of node 76 also includes the nodes 81 and 79, which are not alerted. These two nodes know that their neighbor 76 is not the attacker. If they share this information with the nodes in the alerted region, they can help them to distinguish the attacker.

The external ring reinforcement phase is initiated by the nodes in the alerted region. They broadcast a request to their non-alerted neighbors, including the set $\mathcal{P}$ in the message. The intended receivers check to see if any nodes in $\mathcal{P}$ are their neighbors and broadcast a message voting *in favor* of them.

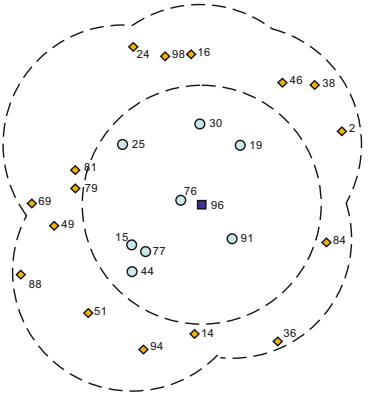


Fig. 5. The attacker's external ring is defined by the nodes which are 2-hops away from the attacker.

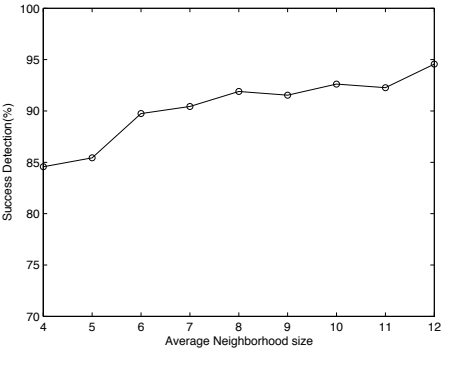


Fig. 6. The overall success rate of the simulated intrusion detection protocol for different neighborhood sizes.

The response message sent by any node of the external ring is forwarded by alerted nodes as in previous phases, such that it can reach all nodes in the alerted region. It is also signed using the next key in the key chain of the sender. The key is released after some fixed period of time and used by the receivers to authenticate the message.

6 Simulation Results

We simulated a sensor network of 100 nodes placed uniformly at random in order to test our intrusion detection algorithm. Figure 6 shows the probability that the IDS system successfully identifies the attacker. To calculate it we run the simulation in 10,000 different topologies, choosing each time a random attacker. If the voting phase was conclusive the protocol ended, otherwise the external ring reinforcement phase was activated. As the subsequent analysis showed, the cases where the protocol did not succeed were all due to the fact that IDC or NC were not satisfied in the given situation.

7 Implementation

In this section, we present experimental results from our implementation of the proposed IDS algorithm. The goal is to exhibit a reference implementation and check the feasibility of an IDS without focusing on its efficiency. Even so, the result shows that such a system is lightweight enough to be a viable and realistic solution from the real deployment perspective. Moreover, various efficiency improvements of the implementation are possible. We leave them to future work.

7.1 Memory and Computational Requirements

The memory footprint of the our implementation is an important measure of its feasibility and usefulness on memory constrained sensor nodes. Table 1 lists the memory footprint of the modules, compiled for the MSP430 microcontroller.

The largest module in terms of RAM footprint is the Key Management module. Its size depends on the maximal number of node's neighbors which is configurable and currently set to 8. For each neighbor, a 10-byte key must also be stored. In terms of ROM, the largest module is the Voting module, since it

Table 1. Size of the compiled code, in bytes

Module	RAM	Code Size
Neighborhood Discovery	136	968
Exchange of Keys	184	3628
Reliability (ADV-REQ)	104	32
Voting	159	4844
Total	583	9472

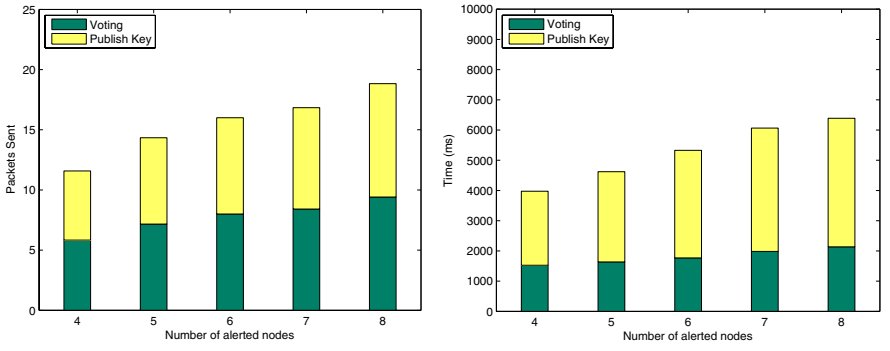


Fig. 7. (a) Measured communication cost for different number of alerted nodes. (b) Detection time for different number of alerted nodes.

has the most lines of code. In total, the IDS leaves enough space in the mote's memory for user applications. For example, Tmote Sky has 10 KB of RAM and 48 KB of program memory.

7.2 Experiments

To evaluate the performance of the implementation of the IDS, we tested it in a real environment. We deployed several nodes in random topologies on the floor of an office building, set a node to be the "attacker" and gradually incremented the number of its neighbors to form larger alert regions. For each alert region size, we repeated the experiment for 20 random topologies.

The experiments were performed with motes running a typical monitoring application. We loaded the Delta application, where the motes report environmental measurements to the base station every 5 seconds. We also deployed the MultihopLQI protocol at the routing layer, which is an updated version of the MintRoute protocol [12] for the Chipcon CC2420 radio. We tuned it to send control packets every 5 seconds. Our goal was to investigate how well the IDS would perform under the presence of traffic on other layers. Then we started an attack to trigger the IDS protocol.

Figure 7(a) depicts the communication cost of the protocol measured in packets sent by a node. In particular, we broke it down to the packets exchanged for the voting phase and the publish key phase (as a total of exchanging the votes, ADV, REQ and keys). As expected, the number of packets exchanged in the two phases is almost the same, since the message dissemination protocol does not change. For small alert region sizes the cost is only about 12 packets, while for more dense regions the cost still remains low (19 packets). This is the total communication cost per attack and involves only the nodes in the alert region. The number of packets depends on the topology and the number of the alerted nodes, which determine the number of votes and keys circulated amongst them.

Next we measured the time that each phase of the IDS protocol required, i.e., the voting phase and the publish key phase. Figure 7(b) shows the measured mean times for each of the above phases, for different number of alerted nodes (i.e., attacker's neighborhood). We can see that the time for the voting phase increases only moderately as the number of alerted nodes increases, and contributes the smallest overhead in the total delay. The most time-consuming phase is the publish key phase, where nodes exchange their keys and verify the votes. To get a better insight, we measured the time needed for the computational operations within this phase. It turns out that the computations (key verification, validation of the signatures on the votes, and the construction of the final result) are responsible only for about 30% of the consumed time, whereas the communication is responsible for the rest.

8 Conclusions and Future Work

In this paper we made a first attempt to formalize the problem of intrusion detection in sensor networks, and showed the benefits and theoretical limitations of the cooperative approach to intrusion detection. We presented necessary and sufficient conditions for successfully exposing the attacker and a corresponding algorithm that is shown to work under a general threat model.

Our investigation of the case of a single attacker ($t = 1$) gave very valuable insights into the solvability of cooperative intrusion detection. In future work we plan to concentrate on the case where the attacker can capture more nodes ($t > 1$). We also plan to look into dynamic neighborhood changes, in particular, into secure node addition and removal in sensor networks.

The intrusion detection algorithm we discussed is the first *generic* algorithm for intrusion detection in sensor networks. Although individual solutions to specific problems might be more efficient, our reference implementation demonstrates that our algorithm is lightweight enough to run on sensor nodes. Thus, studying the problem of intrusion detection in sensor networks is a viable research direction and with further investigation it can provide even more attractive solutions for securing such types of networks.

References

1. Bhuse, V., Gupta, A.: Anomaly intrusion detection in wireless sensor networks. Journal of High Speed Networks 15(1), 33–51 (2006)
2. da Silva, A.P., Martins, M., Rocha, B., Loureiro, A., Ruiz, L., Wong, H.C.: Decentralized intrusion detection in wireless sensor networks. In: Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks (Q2SWinet 2005), pp. 16–23. ACM Press, New York (2005)
3. Kargl, F., Klenk, A., Weber, M., Schlott, S.: Sensors for detection of misbehaving nodes in MANETs. In: Flegel, U., Meier, M. (eds.) Detection of Intrusions and Malware & Vulnerability Assessment, GI SIG SIDAR Workshop, DIMVA 2004, Dortmund, Germany. LNI, vol. 46, pp. 83–97. GI (2004)

4. Krontiris, I., Dimitriou, T., Freiling, F.C.: Towards intrusion detection in wireless sensor networks. In: Proceedings of the 13th European Wireless Conference, Paris, France (April 2007)
5. Krontiris, I., Dimitriou, T., Giannetsos, T., Mpasoukos, M.: Intrusion detection of sinkhole attacks in wireless sensor networks. In: Proceedings of the 3rd International Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors 2007), Wroclaw, Poland (July 2007)
6. Kulik, J., Heinzelman, W., Balakrishnan, H.: Negotiation-based protocols for disseminating information in wireless sensor networks. Wireless Networks 8(2/3), 169–185 (2002)
7. Lamport, L.: Password authentication with insecure communication. Communications of the ACM 24(11), 770–772 (1981)
8. Loo, C.E., Ng, M.Y., Leckie, C., Palaniswami, M.: Intrusion detection for routing attacks in sensor networks. International Journal of Distributed Sensor Networks (2005)
9. Mishra, A., Nadkarni, K., Patcha, A.: Intrusion detection in wireless ad hoc networks. IEEE Wireless Communications 11(1), 48–60 (2004)
10. Onat, I., Miri, A.: An intrusion detection system for wireless sensor networks. In: Proceeding of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, vol. 3, pp. 253–259 (2005)
11. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)
12. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: SenSys 2003: Proceedings of the 1st international conference on Embedded networked sensor systems, pp. 14–27 (2003)

SCOPES: Smart Cameras Object Position Estimation System

Ankur Kamthe, Lun Jiang, Matthew Dudys, and Alberto Cerpa

Electrical Engineering and Computer Science
University of California, Merced CA 95343, USA
{akamthe,ljiang2,mdudys,acerpa}@ucmerced.edu

Abstract. Wireless camera sensor networks have to balance the conflicting challenges imposed by the detection performance, latency and lifetime requirements in surveillance applications. While previous studies for camera sensor networks have addressed these issues separately, they have not quantified the trade-offs between these requirements. In this paper, we discuss the design and implementation of SCOPES, a distributed Smart Camera Object Position Estimation System that balances the trade-offs associated with camera sensor networks. The main contribution of the paper is the extensive evaluation of parameters affecting the performance of the system through analysis, simulation and experimentation in real-life conditions. Our results demonstrates the effectiveness of SCOPES, which achieves detection probabilities ranging from 84% to 98% and detection latencies from 10 seconds to 18 seconds. Moreover, by using coordination schemes, the detection performance of SCOPES was improved with increased system lifetime. SCOPES highlights that intelligent system design can compensate for resource-constrained hardware and computationally simple data processing algorithms.

1 Introduction

In the past decade, the day-to-day life of every human has been invaded by a plethora of sensors. Networks comprised of large numbers of these sensors have been deployed to gather data from the surrounding environment. From simple inexpensive photo sensors to complex camera sensors, the acquisition of data has grown easier but has imposed a greater challenge on the data-processing side. In general, the data processing software is comprised of image processing algorithms for feature extraction and interpretation. Techniques like Viola-Jones [1] build a classifier from a large set of potential features to detect faces with a very high detection rate and provide results in real-time on platforms with high processing power (384x288 pixel images at 15fps on a 700MHz CPU). However, an implementation of the Viola-Jones algorithm provided with the CMUcam3 platform (60MHz with no floating point capability) takes about 5-6 seconds to detect faces [2]. Due to high computational complexity, these techniques do not find widespread adoption in resource-limited wireless sensor networks.

Wireless sensor network-based detection and tracking systems use either computationally lightweight algorithms when performing in-node processing of data

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 279–295, 2009.
© Springer-Verlag Berlin Heidelberg 2009

or data aggregation techniques when transmitting raw data for centralized processing. Aslam et al. [3] describe a centralized framework for tracking a moving object by using a particle-filter for a binary sensor network. VigilNet [4] is a detection and classification system for tracking metallic objects using information from multiple magnetometer and acoustic sensors. In such approaches, the use of simple sensors for tracking entails dense deployment to localize an object or to infer its direction information.

In contrast, using camera sensor networks, we can infer position and direction information from a single sensor by identifying an object and tracking its position in successive images, respectively. This reduces the density of nodes required to gather data while improving the data fidelity manyfold. However, camera sensors are not without their caveats; namely, network lifetime, latency and detection performance. The power consumption of wireless radios transmitting high bandwidth image/video data to a central base station would result in rapid decrease in the lifetime of the sensor mote, in addition to causing network congestion. In-node computations process the raw data into summarized meta-data, which reduces the amount of data to be transmitted. However, this introduces latency because of limitations in the computational capabilities of sensor nodes. This necessitates, to reduce latency, the usage of lightweight processing algorithms and, to avoid missed detections, the redistribution of sensing over multiple sensors. At the same time, the low complexity of the data processing techniques should not compromise the detection requirements of the application.

Our goal is to improve the performance of wireless sensor network-based surveillance systems with synergy between the underlying hardware and software infrastructure. The paper aims to show that SCOPES can achieve comparable, if not better, detection performance by intelligent utilization of resources and computationally lightweight algorithms in a resource-constrained environment, while ensuring long lifetimes. Previous studies [5,6,7] have addressed algorithmic issues for camera sensor networks but provide limited performance evaluation (see Section 5). In this paper, we do not concentrate on the development of new image processing approaches for camera sensor networks. Instead, we provide a much more complete and extensive performance evaluation in a real-world environment and design and implement solutions to the network lifetime, latency and detection quality issues affecting the performance of any camera sensor network. The design of SCOPES incorporates simple but fast image processing algorithms for background update and object recognition, thereby reducing the amount of data transmitted and the processing latency. The redundancy in sensor deployment is harnessed to implement a distributed coordination algorithm to handle node failures. SCOPES was evaluated in real world conditions, with results being compiled from data of a real-life deployment of a camera sensor network system for counting people in different sections of a building. We present results quantifying the trade-offs between data processing latency, memory usage, power consumption and detection performance for our system. To complete the discussion, the paper provides a comparison of SCOPES with closely related works in wireless camera sensor networks for surveillance purposes (see Section 5).

2 System Description

2.1 Hardware and Software Infrastructure

Our SCOPES implementation comprises of an Cyclops camera [8] interfaced with a Moteiv Tmote Sky module via an intermediate adapter board. The Cyclops consists of an ADCM-1700 imager, 512KB of external SRAM and a 4MHz ATmega128L micro-controller (MCU). As the MCU can address a maximum of 64KB of memory, the external SRAM is divided into eight, 64KB memory banks ($nBanks$, from Table 3). The Cyclops captures 10 64x64 pixel grayscale images per bank (i.e., 80 total). It performs local detection and processing of the iamges and sends summarized data (see Section 2.2: Object Recognition) to the Tmote module which routes it to the base station using multihop communication. The Cyclops and the Tmote run the TinyOS operating system (see Figure 1).

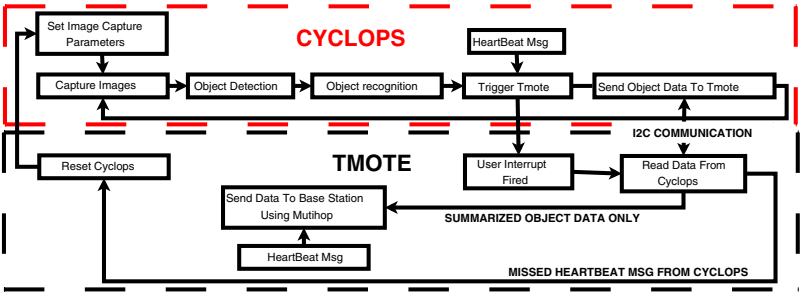


Fig. 1. Software Block Diagram of SCOPES: the arrows indicate the logical sequence of operations and interactions between the two devices

2.2 Algorithms

Object Detection (OD): The goal of object detection is to determine the presence of an object in the image foreground, if any, and to update the background. In order to achieve low processing latency, a modified background subtraction algorithm is implemented for object detection. After background subtraction, we use two preset thresholds (OBJECT-THRES = 40 and SHADOW-THRES = 15) to assign a label (OBJECT, SHADOW or BG) to each pixel (see Figure 2). Depending on the label assignment, the value of the corresponding background pixel is updated using an Exponentially Weighted Moving Average (EWMA). In the EWMA, we give less weight to the value of the current pixel if it is classified as an object or shadow, as these changes to the background are less likely to be of a long term nature. The weights are preset based on the image capture speed of the camera (see Table 1), the area covered by the camera 2.4m x 2.4m) and the speed of objects (approx. 1.2 m/s). In the current implementation, an object would need to be immobile for atleast 5s before being classified as a background pixel. This way, temporal background changes will be assimilated into the background over time. When the number of pixels labelled as OBJECT

```
count = 0
for (each pixel i in current image) do
    delta = | img(i) - bg(i) |
    if (delta ≥ OBJECT-THRESH)
        pixel(i) = OBJECT
        bg(i) = 0.99 * bg(i) + 0.01 * img(i)
        count++
    else if (delta ≥ SHADOW-THRESH)
        pixel(i) = SHADOW
        bg(i) = 0.95 * bg(i) + 0.05 * img(i)
    else
        pixel(i) = BG
        bg(i) = 0.85 * bg(i) + 0.15 * img(i)
```

Fig. 2. Pseudo-code for background subtraction and update

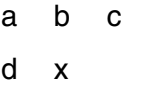


Fig. 3. Grouping Pixels

Table 1. Time (in second) for executing various image processing operations $nFrames$ images at a time on the Cyclops (Note: averaged over 100 cycles)

Cyclops Action	nFrames 1	5	10
Image Capture (IC)	0.68	1.43	2.51
IC w/ OD	0.7	1.54	2.74
IC w/ (OD + OR)	1.3	4.71	9.50

Table 2. Basic algorithmic rules for group assignments for pixel x. (Note:− indicates group id is not assigned.)

Pixel b	Pixel d	Action
-	-	new group id
G_1	-	group G_1
-	G_1	group G_1
G_1	G_2	group G_1 (merge G_2 in G_1)

exceed a preset threshold, an object detection is signalled. In the case of an object detection, pixels labelled SHADOW are relabelled as OBJECT if they have at least one neighbor that is an OBJECT pixel.

Object Recognition (OR): In this step, we try to group all pixels labelled as OBJECT by raster scanning the image starting from the top left corner. A pixel is considered to be part of an object if its left, top, top-left and top-right neighbors are labelled OBJECT. For example, in Figure 3, let x be the pixel in consideration and a, b, c and d are its top-left, top, top-right and left neighbors, respectively. If x, a, b, c and d are all labelled OBJECT, then x is considered to be part of an object. Assigning group id's is done using the rules explained in Table 2. Small groups in close proximity of each other are merged to account for fragmentation of a big object. For each object, information regarding the centroid (x and y coordinates), the number of pixels and the number of consecutive frames in which the object is detected, is maintained.

Direction Inference: In SCOPES, the nodes were deployed in hallways to detect the transitions of people between different sections of a building floorplan (see Section 3.3). This entailed determining movement of people in only two directions. In order to infer direction, objects in successive frames are matched according to their size (in pixels). Any object in the current frame that cannot be matched to another object in the previous frame is stored as a new object. Information for objects from previous frames that disappear or cannot be matched to objects in current frame are saved into a data structure.

Table 3. Notations used in the paper with the associated meanings

Term	Explanation
$nFrames$	number of images captured consecutively
$nBanks$	number of memory banks
$T_S^{nFrames}$	cyclops IC time for $nFrames$ images
T_S	total cyclops IC time (camera ON), depends upon $nBanks$
T_{OD}	(avg.)object detection time for $nBanks \times nFrames$ images
T_{OR}	(avg.) object recognition time per image
P	power consumption per node
DP	detection probability per node
DFP	detection failure prob. per node, $1 - DP$

Information regarding the original position, displacement and number of consecutive frames is maintained for each object that appears across successive images in the current memory bank. After processing all the images in the current bank, an array of data structures containing information on a maximum of four objects is transferred to the base station via the Tmote.

Density Estimation Algorithm: On the base station, the packets coming from the various nodes are deconstructed. Messages are classified by source (node id) and time of occurrence. From the object data in the Cyclops payload, we can infer the direction of motion from the initial position and the displacement vector. Since, we have prior information about the deployment of the nodes, counting the transitions of objects enables the base station to compute the distribution of people in different sections of the building over time (assuming some initial distribution). Objects with no direction information are filtered out.

3 Performance Evaluation of SCOPES

In this section, we present results quantifying the trade-offs between data processing latency, memory usage, power consumption and detection performance. Table 3 shows the notations for the parameters used in the discussion sections.

3.1 Objective Functions

Our goal for SCOPES was to function as a surveillance system to monitor the occupancy of indoor environments such as office buildings. Some metrics and objective functions of interest are as follows:

Global/Local Density Estimate: How accurately can we estimate the occupancy of each section and of the total area covered?

Power Consumption: What is the system lifetime when powered by batteries?

Memory Usage: How much memory is required to achieve acceptable performance? How is the performance affected by memory size?

Detection Latency: How long does the system take to report data?

Detection Probability: How good is the estimate of the movement of people across different sections in the building?

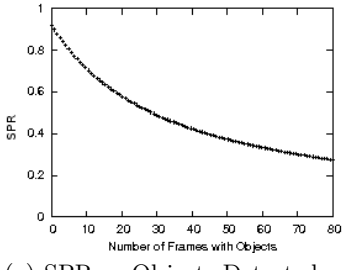
3.2 Simulation and Analysis

Since people passing under a camera can be regarded as a Poisson process, we model their inter-arrival times with an exponential distribution as follows:

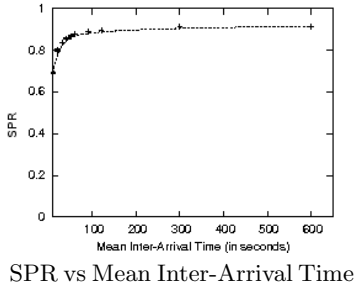
$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} , & x > 0 \\ 0 , & x < 0 \end{cases}$$

where $\frac{1}{\lambda}$ is the mean inter-arrival time for an event. The number of frames in which an object appears is modeled by a uniform distribution (min=2;max=10) to account for variation in speed of people. We simulate the operation of a SCOPES node in the GNU R statistical computing package.

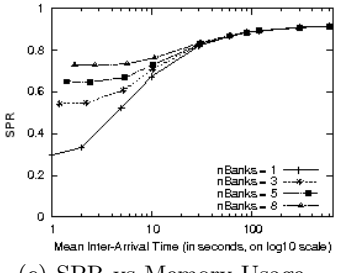
Sensing-Processing Ratio (SPR): In SCOPES, the camera is not operated in trigger-driven or schedule-driven modes discussed in previous studies [7]. Instead, each camera is either in active period, capturing and processing images, or in idle period, wait interval between successive active periods. *In this paper, we refer to the ratio of time taken to capture images by the camera and the total active period i.e., the sum of the image capture and processing times, as sensing-processing ratio.* In our discussions, the sensing-processing ratio is the penalty incurred by the system as a result of the data processing latency. A high sensing-processing ratio is an indicator of lower data processing latency and vice-versa.



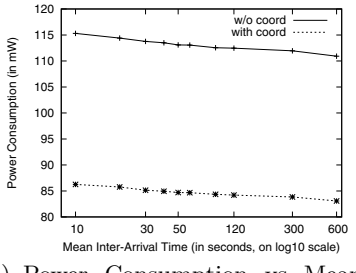
(a) SPR vs Objects Detected



(b) SPR vs Mean Inter-Arrival Time



(c) SPR vs Memory Usage



(d) Power Consumption vs Mean Inter-Arrival Time

Fig. 4. (a) shows the variation in SPR of a node as a function of the number of image frames containing an object. (b) shows the SPR as a function of the mean inter-arrival time. (c) shows the change in SPR as a function of the mean inter-arrival times for different memory usages. (d) shows Power Consumption as function of the mean inter-arrival times.

There are two main reasons for operating the camera as described above; first, the camera hardware in current sensor networks does not allow concurrent image capture and processing of data and second, the object recognition algorithm introduces long latencies between successive image capture periods. Understanding sensing-processing ratio is important, since it affects many of our objective functions, including global/local position estimation, power consumption, detection probability and detection latency.

In Figure 4(a), we observe the variation in SPR of a node with respect to the total number of images N in which an object (person) is detected. This relationship can be expressed as follows:

$$SPR = \frac{T_S}{T_S + T_{OD} + N \times T_{OR}} \tag{1}$$

Background subtraction needs to be performed for all the images resulting in a fixed cost T_{OD}. The object recognition function needs to be executed on only the N frames in which an object is detected. As object recognition incurs the highest processing cost (T_{OR}, see Table 1), the SPR of a node is affected by the time taken for object recognition in each image. During the image processing latency period, an object passing beneath the camera will be missed as the Cyclops is not capable of simultaneously capturing and processing images. Thus, a low SPR will result in lower detection probability.

Figure 4(b) shows the variation in SPR with respect to the mean inter-arrival time $\frac{1}{\lambda}$ of an object detected by the camera with fixed amount of memory ($nBanks = 8$ and $nFrames = 10$, refer Section 2.1 for cause of $nBanks$). This relationship can be expressed as follows:

$$SPR = \frac{nT_S}{n(T_S + T_{OD}) + \sum_{i=1}^{n} \alpha_i \beta_i T_{OR}} \tag{2}$$

where n is the number of times we capture a set of 80 images, α_i is the number of times an object is detected and β_i is the average number of frames occupied by the object in the current (i^{th}) set of images. When the mean inter-arrival time is low, more objects are detected and the camera spends a longer time processing the image data, leading to a low SPR. Hence, longer data processing time leads to lower detection probability as the camera cannot capture images during that period.As the inter-arrival time increases, the SPR increases because the relative proportion of image processing time decreases.

In Figure 4(c), we observe the variation in SPR with respect to the mean inter-arrival time as a function of memory usage (varying $nBanks$). The amount of available memory dictates the space available to store images captured in time T_S ($=nBanks \times T_S^{nFrames}$). This relationship can be expressed as:

$$SPR = \frac{n(nBanksT_S^{nFrames})}{n(nBanksT_S^{nFrames} + T_{OD}) + \sum_{i=1}^{n} \alpha_i \beta_i T_{OR}} \tag{3}$$

As the mean inter-arrival times varies from low to high, the SPR increases with memory usage since a node captures more images while spending a lower percentage of its time processing the image data. Hence, in general, more memory

leads to a better SPR. For long mean inter-arrival times, the amount of memory does not have a significant effect on the SPR of a node.

Power Consumption (P): The lifetime of battery powered sensor nodes is directly affected by the power consumption of the system. The power consumed by the Cyclops and the Tmote Sky in different modes of operation is given in Table 4. The relationship between the power consumption and the different modes of operation of the node can be expressed as follows:

$$P = P^{sensing}_{cyclops} + P^{proc}_{cyclops} + P^{sleep}_{cyclops} + P^{RX}_{tmote} + P^{TX}_{tmote} \quad (4)$$

We analyse the power consumption under two different scenarios for node operation: (i) without coordination (multiple nodes sense the area at the same time) and (ii) with coordination (multiple nodes sense the area in non-overlapping intervals of time, see Section 4 for node coordination scheme details). Figure 4(d) shows the variation in power consumption for each of 3 nodes deployed to sense an area as a function of the mean inter-arrival time of an object.

For case (i), the Cyclops on each of the nodes is capturing and processing data all the time, i.e., there are no idle periods (Sleep mode). The power consumed by the Cyclops is slightly higher when it is processing image data stored in the external SRAM as compared to the power consumed in Image Capture mode (refer Table 4). This leads to higher power consumption when the inter-arrival time is low as the Cyclops spends a higher proportion of its active time processing data. For case (ii), with sensing coordination between the three nodes, the power consumed by each node is significantly lower than in the first case as the Cyclops has idle periods while waiting for its turn to sense the area.

Detection Failure Probability (DFP): Detection failures occur in the form of false negatives and false positives. However, we define Detection Failure Probability (DFP) as the probability that none of the camera nodes covering a section report the presence of a person passing under the camera. DFP quantifies the effect of *only false negatives* on our system. False negatives mainly

Table 4. Power Consumption of the Cyclops and Tmote Sky Modules. (For more details, refer to [8] and the Tmote Sky data sheet.)

Device Operation	Notation	Power
Cyclops		
- Image Capture	$P^{sensing}_{cyclops}$	42mW
- Extended Memory Access	$P^{proc}_{cyclops}$	51.5mW
- Sleep	$P^{sleep}_{cyclops}$	0.7mW
Tmote		
- MCU + Radio RX	P^{RX}_{tmote}	65.4mW
- MCU + Radio TX (0dBm)	P^{RX}_{tmote}	58.3mW

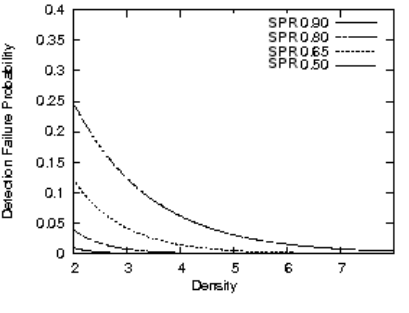
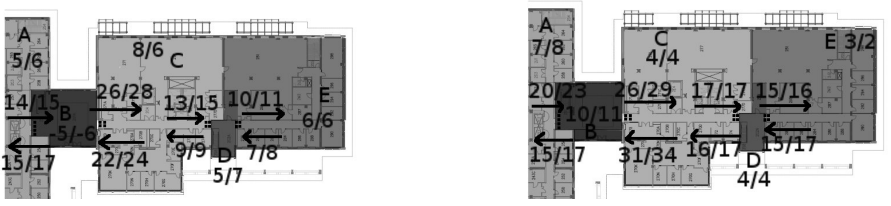


Fig. 5. Detection Failure Probability as a function of density of nodes covering the same area and SPR



(a) Morning Map (b) Noon Map

Fig. 6. Occupancy and Transition Maps. The arrows indicate the direction of motion. The different sections of the floorplan are shaded and labelled with different alphabets. The numbers indicate counts from SCOPES (left) and from ground truth (right).

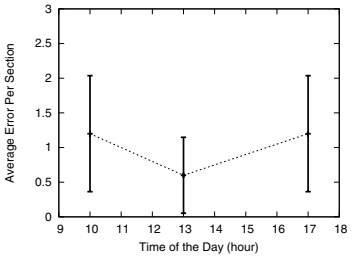
depends on the SPR and to a lesser extent on the static thresholds in the object detection algorithm. Detection failures, due to static thresholds, are difficult to simulate as they might not provide an accurate representation of real-life conditions. We only analyse the relationship between DFP and multiple nodes n sensing an area at the same time (sensing without coordination) as a function of varying SPR. This relationship can be expressed as:

$$DFP = (1 - SPR)^n \tag{5}$$

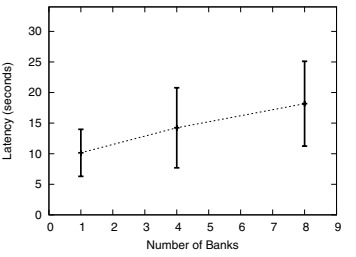
Figure 5 shows that the DFP decreases with higher SPR and number of nodes n. Since, the worst case processing time is bounded because of memory constraints, the SPR cannot fall below a certain number. Having coordination among the nodes will improve detection failure probability by eliminating the chances of a missed detection due to SPR. However, there will still be some missed detections because of the deficiencies of the hardware and software in the underlying platform (See Section 4.3).

3.3 Experimental Deployment

We deployed 16 nodes on the ceiling of the corridors in an office building. The deployment of nodes was done in this fashion to reduce privacy concerns of individuals by sensing in the common areas of the floorplan. The floorplan was separated into 5 sections by deploying the nodes in groups at transition points. *In each group, multiple nodes sense the same area at the same time, i.e., operating without coordination.* For collecting the ground truth, we installed two Panasonic KX-HCM280A network web cameras to record the movement of people. They are capable of capturing 10 frames per second (fps) at a resolution of 640×480 pixels. These images are timestamped using an NTP synchronized machine. The ground truth data is processed using haar cascades implemented in the OpenCV library [9] to provide a list of images in which a human being is detected. We manually corrected the OpenCV output for the false positives and false negatives in the processed ground truth. For computing detection probability and latency, we compare the manually processed and corrected ground truth data with the



(a) Position Estimation Error vs Time of the Day



(b) Detection Latency vs Memory Usage

Fig. 7. Fig. 7(a) shows the Average Position Estimation Error (number of people) per section for different times of the day. Fig. 7(b) shows the Detection Latency as a function of the number of memory banks.

data collected from the SCOPES logs. The following is a list of experiments that we conducted for the performance evaluation:

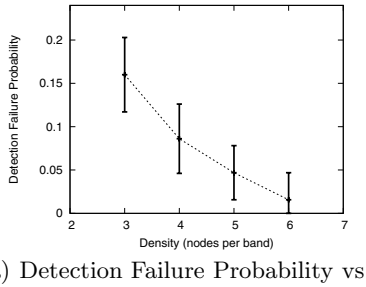
Occupancy and Flow Estimation: 4 groups of nodes with 4 nodes in each group were deployed ($nBanks = 8$, $nFrames = 10$). The experiment was conducted twice for different two-hour periods of the day

Memory Usage and Detection Latency: 3 groups of nodes with 4 nodes in each group were deployed for each value of $nBanks$ ($nFrames = 10$).

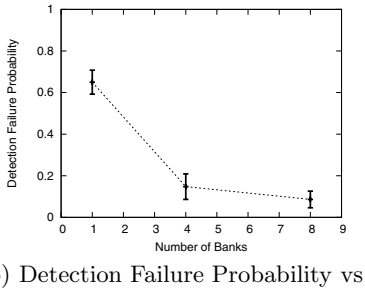
Detection Probability: experiment repeated thrice with 3, 4, 5 and 6 nodes deployed in each group ($nBanks = 8$, $nFrames = 10$).

3.4 Experimental Results

Density Estimation: The first aspect we address is the evaluation of SCOPES for building density estimation maps of area occupancy by counting the



(a) Detection Failure Probability vs Density of Nodes



(b) Detection Failure Probability vs Memory Usage

Fig. 8. Fig. 8(a) shows Detection Failure Probability as a function of the density of nodes covering the same area. Fig. 8(b) shows the Detection Failure Probability as a function of the number of memory banks.

transitions across the different sections. Figure 6 shows the occupancy and transition estimation maps created from data acquired from individual SCOPES experiments. From our results, we see that, we are able to track the movement of people over more than 73 sq. meters of the Engineering Building with a small, reasonable error. Figure 7(a) shows the average density estimation error for all the sections at different times of the day. Since, the error bars are overlapping with the mean values, we can say that there is no statistically significant change in average error which remains bounded (under 2) at different times of the day. The result shows that with suitable number of nodes and deployment location, embedded camera sensor networks such as SCOPES can provide adequate performance for density estimation purposes in real-world scenarios.

Detection Latency: Detection Latency is the time it takes for a node to report a person transitioning among different areas to the base station. In the Cyclops, the 512KB of available memory is partitioned into eight 64KB banks. Each node first captures $nBanks \times nFrames$ images and then it starts processing the image data in each bank. When the Cyclops is able to infer direction for an object from images in a certain bank, it will transfer the summarized object data to the Tmote for that bank. The Tmote routes the data to the base station via the radio. In our experiments, the base station was located 2 hops away from the farthest group of nodes. Figure 7(b) shows the variation in detection latency as a function of the number of memory banks used for storing images. In Figure 7(b), we observe that the detection latency is directly proportional to the amount of memory utilized for storing images. The detection latency is 10 seconds when $nBanks = 1$. It increases to 18 seconds when $nBanks = 8$. As the amount of available memory ($nBanks$) increases, the Cyclops can capture a higher number of images before processing the image data, resulting in longer detection latency. This shows for camera sensor networks that capture sequences of images before processing them, storing more image data can increase the detection latency, which could adversely affect the responsiveness of the surveillance system.

Detection Failure Probability (DFP): Figure 8(a) shows the variation in DFP i.e., false negatives as a function of the number of nodes used to cover an area. In general, deploying more nodes improves the DFP which agrees with our simulation results (see Figure 5). However, beyond 6 nodes we do not see an improvement in DFP because of the limited capabilities of our nodes. In Figure 8(b), we see the variation in DFP under memory constraints. In these experiments, we vary the number of memory banks used for storing image data. Figure 8(b) shows that DFP gets significantly reduced as we increase the number of memory banks. From Figures 8(b) and 4(c), we confirm that as the numbers of memory banks increases, SPR increases, leading to lower DFP.

Our experimental evaluation highlights and quantifies the trade-off between detection latency and detection performance as a function of memory usage and node density. For camera sensor networks like SCOPES, increasing the number of nodes would keep the detection latency low and improve detection performance at the expense of increased deployment cost. Also, by deploying sufficient number

of nodes to cope with worst case SPR, we can enable lower detection latency and hence, a more responsive system.

4 Improving SCOPES Using Node Coordination

To ameliorate the effects of concurrent data processing latency periods for nodes working in a uncoordinated manner, we decided to implement a scheduling and coordinated sensing scheme. The goal of the scheme is to improve the performance of the existing system by reducing the detection failure as well to decreasing the power consumption of the nodes (refer Section 3.2). The design requirements for our node coordination scheme are two-fold: (1) Nodes covering the same area or nodes in close proximity should provide near-continuous sensing coverage for the area, and (2) Nodes should provide near-continuous sensing coverage for an area even if some nodes stop functioning.

4.1 Clustering Algorithm

The clustering algorithm (see Figure 9) executes periodically once every hour. At the beginning, the nodes change their RF power level to reduce the transmission distance ($RF2$). Each node then starts a one shot timer (Timer1) with a random interval up to a maximum of T1 seconds. After the Timer1 fires, a node sends a $GROUP\_ASSOC$ message declaring itself as the group head. It then starts Timer2 with an interval of T2 seconds. All nodes that are within close proximity of the group head respond by sending a $GROUP\_INFORM\_CH$ message. After Timer2 fires, the group head broadcasts a message containing information regarding the group head and the associated group members. Since, these radio messages do not propagate beyond a certain distance, we ensure that nodes in

```
INITIALIZE()
    Change RF Power Level to RF2.
    Set Timer1 to fire after a random interval
TIMER1.FIRED()
    If no GROUP_ASSOC packet received,
        broadcast a GROUP_ASSOC message with group head = current node and set isCH = TRUE
        Set Timer2 to fire after a specific interval
TIMER2.FIRED()
    Broadcast a GROUP_INFORM_MEM message with
    information such as group head and other group members.
RECEIVEMSG.RECEIVE()
1   GROUP_ASSOC message received,
        isCh = FALSE
        send GROUP_INFORM_CH message to associate with a group head
2   GROUP_INFORM_CH message received,
        associate sender node id as part of group
        If Timer2 is not running, set Timer2 to fire after a specific interval
3   GROUP_INFORM_MEM message received,
        copy group data from packet (group head and other members information)
```

Fig. 9. Grouping Algorithm Pseudo-Code

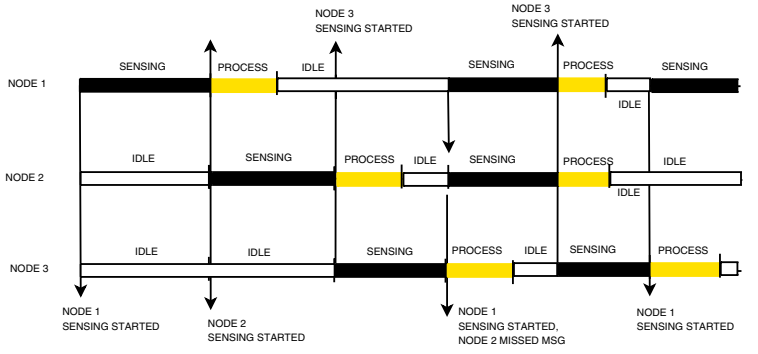


Fig. 10. Illustration of the Group Coordination Algorithm

close proximity are part of the same group. This approach will work if the distance between groups of nodes is greater than the radio propagation distance for the set RF power level. For SCOPES, we empirically set T1=8s and T2=60s. The RF power level for group communication was set to $< -25dBm$.

4.2 Distributed Coordination and Scheduling

Once groups are formed, a node coordination scheme enables non-overlapping, continuous sensing coverage. Our notion of node coordination uses "soft-state" [10] to achieve continuous sensing coverage for a particular area. An illustration of the working on the scheme is shown in Figure 10. Here, each node sends an update message to its group members before it starts sensing. When the other nodes in the same group receive an update message, they advance their "start sensing" timers by one sensing period. When the sensing timer expires, another node sends an update message informing that it has started sensing the area. This way, we can achieve continuous sensing of an area, given sufficient deployment. As all the coordination messages are sent over the radio, we can never discount the possibility that an update message was missed by a particular node. By design, if an update message is lost, the system does not break down. In the event of a lost update message, nodes are expected to start sensing when their sensing timer expires. This also helps to provide continuous coverage in the event of node failures. When an update message is lost, multiple nodes will sense the area in that cycle but the schedule is resumed as soon as the new update messages are received in the following cycle. The current scheme adapts to node failures while ensuring that cameras are providing continuous sensing coverage all the time and non-overlapping coverage for a majority of the time.

4.3 Performance Evaluation

We performed experiments to evaluate the performance of the SCOPES when nodes work in coordination, sensing in non-overlapping intervals of time. The results are presented (refer Table 5) for a single, two hour experiment involving

Table 5. Comparison of detection performance with and without node coordination

Operation Mode	DFP/False Negative %	False Positive %	Number of Objects
Without Coordination	20.0%	21%	85
With Coordination	15.3%	18.5%	103

Table 6. Counting the number of occurrences of false positives and false negatives along with their causes (NOTE: 1. Data is acquired from a single, two hour experiment with 3 nodes working together with coordination. 2. Under false negatives, 103 was the total number of people passing beneath the SCOPES nodes. 3. Under false positives, 193 is the total number of messages received at the base station).

Mis-Detections	Reason	Occurrences	Percentage
False Negatives	Object detected at memory bank border	4 out of 103	3.8%
	Object detected at limit of Sensing Range	8 out of 103	7.7%
	Software Inadequacy	4 out of 103	3.8%
False Positives	Objects in background	15 out of 193	7.7%
	Over-counting due to split-objects	21 out of 193	10.8%

3 nodes. We compare the performance of the SCOPES system in the presence and absence of coordination (see Section 3.4). Here, we see that DFP is reduced to 15% when nodes work with coordination as compared to 20% without any coordination. The DFP is also the false negative percentage for the system. DFP shows significant improvement with node coordination and we would like to continue that evaluation in future work. We have shown earlier (see Figure 4(d)) that we can significantly reduce the power consumption by using node coordination. Also, the relationship between detection performance, detection latency and memory usage (see Figures 7(b) and 8(b)) is independent of the presence or absence of coordination and hence, we expect these results to show similar trends. To complete our evaluation of the system, we provide a quantitative analysis of the detection failures in SCOPES.

Analysis of Detection Failures: In Table 6, we enumerate the number of occurrences and their respective percentages along with the associated reason for misdetection, in the presence of node coordination.

False Negatives: We report a false negative when the ground truth indicates that there is an object in the foreground whereas our system reports none.

Objects missed due to software: Under-counting occurs when the object detection algorithm is unable to differentiate the object from the background. This happens because the colors of objects in the foreground do not contrast enough against the background to trigger object detection. Another scenario where under-counting could occur is when the object recognition algorithm (see Section 2.2) merges two objects that are in close proximity to each other.

Under-counting objects due to hardware: This occurs due to the following reasons: (a) SPR of nodes and (b) limitations of the sensing hardware. Detection failures due to SPR are avoided by using node coordination. However, the camera fails to detect

an object due to loss of image data when the camera is switching memory banks. Since, the object is seen in only one frame in each bank, the algorithm would report no direction information as it does not combine information from successive banks. This problem could be resolved if memory was continuous and not split into banks. Objects that move close to the sensing range of the camera are missed because the camera is not able to cover the entire object from its point of view.

False Positives: False positives result mainly, due to the high sensitivity of the simple background subtraction algorithms used to detect the presence of objects in the image foreground from the fixed thresholds. This might be due to over counting of objects in the foreground and camera hardware calibration.

Over-Counting Objects: Over-counting occurs because the object recognition algorithm might split one object into two objects. It also occurs when a foreign object becomes part of the background for a short time.

Camera Hardware: In SCOPES, when the camera starts capturing images, at times, the image at the start of the burst exhibits higher brightness as compared to all the rest due to calibration issues. This behavior of the imager results in false positives. However, the resulting message contains information about an object with disproportionately large number of pixels and no direction information. We neglect such objects when computing the false positives for our system.

As part of future work, we would like to provide detection failure comparisons between the output of OpenCV program, using computationally complex algorithms, and our system.

5 Comparisons with Previous Work

In this section, we compare SCOPES with related work in the area of embedded camera sensor networks on issues like processing algorithms, latency, memory usage, detection probability and evaluation methods.

Kulkarni *et al.* [5] presented the design, implementation and evaluation of SensEye, a multi-tier camera sensor network for surveillance applications. The work aimed at showing that a multi-tier network can balance the conflicting goals of latency and energy-efficiency. In the evaluation experiments, circular objects were projected onto a wall with an area of $3m \times 1.65m$. Objects appeared at different location for a certain time duration with only one object present at a time. SensEye detected 42 out of 50 object appearances. It achieved 100% detection probability when objects are in view for 9 seconds which decreases to 52% when object time duration is 5 seconds. For moving objects, speeds were varied from 0.2m/s (all objects detected) to 0.6m/s (38% objects detected).

As seen in SensEye, a camera-based surveillance system fails when the speed of the object exceeds the capability of the system. From empirical data, it is said that humans move at speeds ranging from 1-1.5m/s [11]. The main difference between the evaluation of SensEye and SCOPES is that SCOPES was evaluated in uncontrolled real-life conditions where it had to account for variations in light

conditions, shadows, occlusions, and the size and speed of people moving in the environment. SCOPES still has an average detection probability of 84% when we deploy 3 camera nodes to cover an area, which improves to 98% for 6 nodes. Based on the image capture speed of the camera, SCOPES will fail to capture information required to detect an object if the object moves at a speed greater than 8m/s (no image data collected) and will fail to infer the direction if the object moves faster than 4m/s (only one image frame collected).

Teixeira *et al.* [6] proposed a motion histogram approach to count people in indoor spaces. The hardware infrastructure comprised of Intel iMote2 sensor nodes with OmniVision OV7649 imagers. The iMote2 sensor platform operates at 104MHz and is capable of processing 8fps while consuming 322mW (Imote + Camera) of power. Six nodes were deployed with minimum overlap between areas covered by the cameras. Each camera has a field of view of 3m x 2m. The experiments consisted of five people moving inside a lab setting.The system has a detection rate of 89.5% when a single person is present inside the camera network. This drops to 82.48% when two people are present and 79.8% for three. Using the case study of the same camera node, Jung *et al.* [7] present lifetime models for trigger-driven and schedule-driven sensor networks. Their models predict the energy budgets under different application requirements. The results in the paper show the variation in the lifetime of the camera sensor network with respect to the detection probability and object inter-arrival rate.

In comparison, in SCOPES, the Cyclops board operates at 4MHz and is capable of processing 1fps while consuming 115mW of power (Tmote + Cyclops). Inspite of the speed of the Cyclops platform (26 times slower than the iMote2) and the simple image processing algorithms, SCOPES is able to achieve detection probability of 84% with 3 cameras which increases to approx. 98% with 6 cameras covering the same area (see Figure 8). On a faster platform such as the iMote2, the SCOPES image processing algorithms would execute in roughly 26ms, eliminating the need for multiple cameras to provide coverage during the detection latency period of a camera while achieving comparable, if not superior, performance to the motion histogram approach. The performance evaluation of SCOPES highlights the point that by using computationally simple image processing techniques it is possible to achieve detection probabilities comparable to techniques like the motion histogram approach, inspite of differences in the computational capabilities of the underlying platforms. In SCOPES we analysed the power consumption for continuous sensing (without coordination) or interleaved sensing (with coordination) nodes, which differ from the operation models considered in [7]. In addition, in SCOPES, we also provide a detailed analysis and evaluation of the memory usage, detection latency and detection probability as a function of the system parameters.

6 Conclusion and Future Work

In this paper, we argued that previous studies lacked extensive performance evaluation of camera sensor networks in real life conditions, raising doubts regarding

the sustainability of such systems. In contrast, through analysis, simulation and extensive experimentation, we showed that deployment of multiple nodes working in coordination with each other eliminates some of the problems associated with network lifetime, data processing latency and quality of detection performance. In addition, we analysed the detection failures of SCOPES by describing the causes of misdetections and quantifying their effects. Our system provides on par or better detection performance than other approaches that have computationally intensive algorithms and more capable hardware, with a slightly higher deployment cost. In summary, the paper presented a comprehensive design for a embedded camera sensor network, alongwith extensive testing of parameters affecting the system. As part of future work, we would like to pursue the development of lightweight image processing algorithms for camera sensor networks. We would also like to investigate the tradeoff between the network traffic and detection quality by sending the entire image instead of summarized data.

References

1. Viola, P., Jones, M.: Robust real-time object detection. International Journal of Computer Vision (2001)
2. Viola Jones Detector for CMUcam3, http://www.cmucam.org/wiki/viola-jones
3. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., Rus, D.: Tracking a moving object with a binary sensor network. In: SenSys 2003, pp. 150–161. ACM Press, New York (2003)
4. Gu, L., Jia, D., Vicaire, P., Yan, T., Luo, L., Tirumala, A., Cao, Q., He, T., Stankovic, J.A., Abdelzaher, T., Krogh, B.H.: Lightweight detection and classification for wireless sensor networks in realistic environments. In: SenSys 2005, pp. 205–217. ACM Press, New York (2005)
5. Kulkarni, P., Ganesan, D., Shenoy, P., Lu, Q.: Senseye: a multi-tier camera sensor network. In: MULTIMEDIA 2005, pp. 229–238. ACM Press, New York (2005)
6. Teixeira, T., Savvides, A.: Lightweight people counting and localizing in indoor spaces using camera sensor nodes. In: ICDSC 2007, September 25-28 (2007)
7. Jung, D., Teixeira, T., Barton-Sweeney, A., Savvides, A.: Model-based design exploration of wireless sensor node lifetimes. In: Langendoen, K.G., Voigt, T. (eds.) EWSN 2007. LNCS, vol. 4373, pp. 277–292. Springer, Heidelberg (2007)
8. Rahimi, M., Baer, R., Iroezi, O.I., Garcia, J.C., Warrior, J., Estrin, D., Srivastava, M.: Cyclops: in situ image sensing and interpretation in wireless sensor networks. In: SenSys 2005, pp. 192–204. ACM Press, New York (2005)
9. OpenCV, http://opencvlibrary.sourceforge.net/
10. Ji, P., Ge, Z., Kurose, J., Towsley, D.: A comparison of hard-state and soft-state signaling protocols. IEEE/ACM Transactions on Networking 15(2), 281–294 (2007)
11. NationMaster Orders of magnitude (speed), http://www.nationmaster.com/encyclopedia/

secFleck: A Public Key Technology Platform for Wireless Sensor Networks

Wen Hu, Peter Corke, Wen Chan Shih, and Leslie Overs

Autonomous Systems Laboratory, CSIRO ICT Centre, Australia
{wen.hu,peter.corke,teddy.wen-chan,leslie.overs}@csiro.au
http://www.sensornets.csiro.au

Abstract. We describe the design and implementation of a public-key platform, secFleck, based on a commodity Trusted Platform Module (TPM) chip that extends the capability of a standard node. Unlike previous software public-key implementations this approach provides E-Commerce grade security; is computationally fast, energy efficient; and has low financial cost — all essential attributes for secure large-scale sensor networks. We describe the secFleck message security services such as confidentiality, authenticity and integrity, and present performance results including computation time, energy consumption and cost. This is followed by examples, built on secFleck, of symmetric key management, secure RPC and secure software update.

1 Introduction

Wireless sensor network (WSN) applications [6,12,1,2,21] are growing. While the importance of security and privacy is generally agreed, it is still largely ignored since the problem is considered impractical to solve given the limited computation and energy resources available at node level. In the future, privacy, authenticity and security will be required for WSN gathered resource utilization (for billing purposes), and authenticity and security of WSN management commands and program downloads. The lesson from the PC industry is that ignoring security at the outset leads to huge pain when the technology becomes ubiquitous.

Symmetric (shared) key algorithms are tractable on mote-class hardware and can achieve message confidentiality. However, key distribution and management remains a significant practical challenge, and these algorithms poorly support message authenticity and integrity. On the Internet, Public Key Cryptography (PKC) is widely used to support symmetric key management, as well as message authenticity and integrity. Researchers have investigated methods to support PK technology in WSN [22,15]. Such approaches have focused on software-based PK technologies, such as Rivest Shamir Adelman (RSA) and Elliptic Curve Cryptography (ECC) but the performance has been poor given the low clock rate and memory availability. Consequently, a smaller RSA public exponent (e) and a shorter key size are chosen, which compromises the security level of asymmetric encryption.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 296–311, 2009.
© Springer-Verlag Berlin Heidelberg 2009

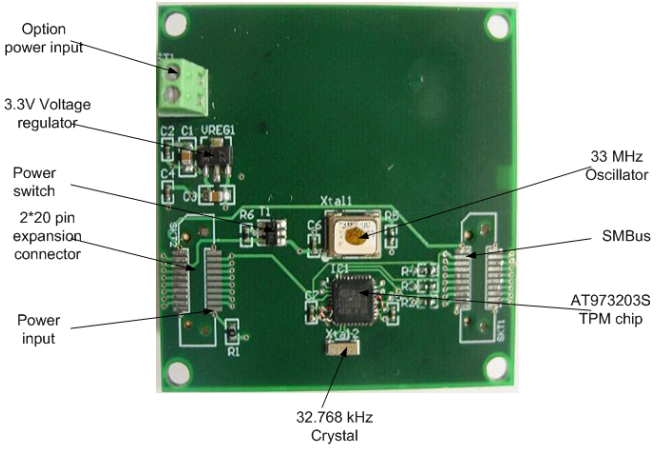


Fig. 1. secFleck TPM module (upper side) — an expansion board for the Fleck WSN node

In this paper, we introduce the design and implementation of secFleck, a PK platform that uses Trusted Platform Module (TPM) hardware to augment the node. Our evaluation shows that the secFleck provides Internet-level PK services with reasonable energy consumption and financial overhead. The contributions of this paper include:

- The design and implementation of the secFleck platform, which includes a standard TPM chip and a set of software primitives, to support Public Key Cryptography (PKC) in a WSN. To the best of our knowledge, the secFleck is the first platform that supports most RSA-based PKC functions (encryption, decryption, signature, and signature verification) in WSN. RSA is the most widely used PKC in the traditional networks such as the Internet.
- Extensive evaluation of the secFleck platform in terms of computation time, energy consumption, memory footprint and cost. The results demonstrate the feasibility of the secFleck platform.
- The demonstration that the secFleck platform is easy-to-use through case studies of how to implement key management, secure software update, and secure Remote Procedure Calls (RPC) services using the secFleck primitives.

The rest of this paper is organized as follows. In section 2, we give a brief overview of the RSA algorithm, which secFleck is based on, followed by a detailed description of the software and hardware architecture of secFleck (Section 3). We evaluate the performance of secFleck in terms of computation time, energy consumption and financial cost in Section 4. Section 5 describes, by means of a case study, how secFleck primitives can be used to implement state-of-the-art key management and secure software update protocol, and improve the protocol's performance. We present related work in Section 6. Finally, we finish with conclusions and future work in Section 7.

2 A Brief Introduction to the RSA Algorithm

In this section, we provide an overview of the Rivest Shamir Adelman (RSA) algorithm [18], which secFleck is motivated by and built upon. We will also discuss some RSA terms and parameters, such as modulus (n), random numbers (p and q), public exponents (e) and key sizes (k), and their implications to the RSA algorithm computation complexity and security levels.

RSA is an algorithm for public key cryptography (PKC), also called asymmetric cryptography, in which the encryption key is different to the decryption key. RSA is the first algorithm that is suitable for signing and encryption, and is used widely in secure communication protocols, such as Secure Shell (SSH) and Secure Sockets Layer (SSL), in the Internet.

The RSA algorithm generates a public key and a private key simultaneously as follows. First, RSA chooses two large random numbers p and q. Second, RSA calculates the product (n) of p and q: $n = pq$, where n is used as the modulus for RSA public and private keys.

Third, RSA calculates the Euler's totient function of n, given by: $\varphi(n) = (p-1)(q-1)$. Fourth, RSA chooses an integer (e, also called public exponent) such that:

$$1 < e < \varphi(n), \tag{1}$$

and

$$gcd(e, \varphi(n)) = 1, \tag{2}$$

where gcd stands for the Greatest Common Divisor (GCD). Public exponent (e) and modulus (n) together comprise the public key.

Fifth, RSA calculates private exponent (d) by

$$de \equiv \ mod\ \varphi(n), \tag{3}$$

where parameters d, p, q are kept secrets.

Since the public key (n, e) of Alice is available to everyone, Bob can then encrypt a plain text message (m) by

$$c = m^e\ mod\ n, \tag{4}$$

where c is the cipher text (cipher) of plain text message m, and $0 \leq c < n$. Only Alice, the owner of kept secrets (d, p, q), can decrypt the cipher (c) and obtain plain text message (m) by

$$m = c^d\ mod\ n. \tag{5}$$

Further, with her private key (d, p, q), Alice can use the RSA algorithm to sign a message by generating a signature (s) by substituting c with a hash value ($H(m)$) of m in Eq. (5). After receiving ($H(m)$, s), Bob uses the same hash function, together with Alice's public key (n, e), to verify the signature by Eq. (4).

Because the sizes of p and q are approximately half of the size of the key size (k), the security level of RSA cryptography is a function of e and k. A popular choice for the public exponent is $e = 2^{16} + 1 = 65,537$. Using small

e values such as 3, 5 or 17 can dramatically reducing computational cost, but will lead to greater security risks [18]. The default e value in secFleck is 65,537. It is common to believe that a RSA key size of 512 bits is too small to use nowadays. Bernstein has proposed techniques that simplify brute-forcing RSA [3], and other work based on [3] suggests that 1024-bit RSA keys can be broken in one year by a device that costs $10 million rather than trillions as in previous predictions [19]. It is currently recommended to use an RSA key at least 2048 bit long. Therefore, the default RSA key size in secFleck is 2048 bits.

3 Platform Architecture

In this section, we discuss both hardware and software modules in secFleck.

3.1 Hardware Module

The core of secFleck is an Atmel AT97SC3203S TPM chip (see Fig. 1) mounted on a Fleck expansion board (see Fig. 2). The TPM chip follows version 1.2 of Trusted Computing Group (TCG) specification for TPM. It has a true Random Number Generator (RNG), which is Federal Information Processing Standards (FIPS) 140-2 compliant. By implementing computationally intensive RSA operations in hardware, the TPM chip performs these operations in an efficient manner. For example, it can compute a 2048-bit RSA signature in 500ms according to the Atmel data sheet.

Fig. 1 is a picture of TPM board. The TPM board, connected to a Fleck, can be enabled to meet WSN application requirements. The TPM board and the Fleck is shown in Fig. 2. The Fleck is a wireless sensor node that features an Atmega 1281 micro controller (8 MHz clock rate and 8 KB memory) and a Nordic nRF905 radio [4]. The TPM module has a 100 kHz SMBus which is similar to the I2C and the TPM is connected to the Fleck's I2C interface. The SMBus makes the TPM chip easily integrated in embedded systems.

Fig. 3 shows a block diagram of the TPM module, which includes the bare minimum of components required for operation: TPM chip, crystal, oscillator, voltage regulator, power switch and an expansion connector.

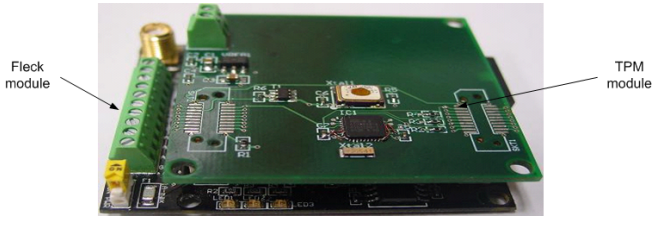


Fig. 2. secFleck (Fleck3 and TPM module)

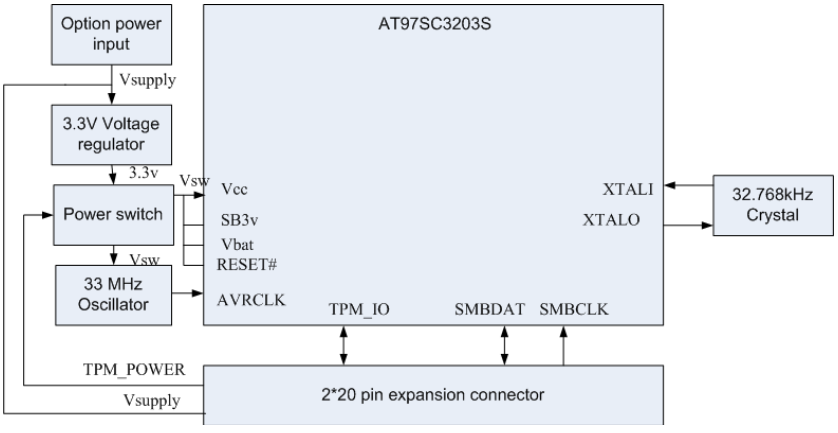


Fig. 3. secFleck TPM module block diagram

3.2 Software Module

For the ease of WSN application developers, we have implemented a set of RSA public key cryptographic primitives as a Fleck OS (FOS) [4] module, which include encryption, decryption, signing, and signature verification etc., as well as XTEA symmetric cryptographic primitives (see Fig. 4). FOS is a C-based cooperative multi threaded operating system for WSN.

Asymmetric Key (RSA) FOS Functions. Previous research [11] shows that the primitives, which allow an application to turn a system component on or off, are important to conserve system energy consumption in sensor networks. secFleck allows applications to duty cycle TPM component by calling primitives `fos_tpm_startup()` and `fos_tpm_turnoff()`. These duty-cycle primitives are more important in secFleck because its current consumption (around 50 mA) is significantly more than Fleck's average current consumption (around 5 mA).

Symmetric keys are typically generated by a pseudo-random number generator in previous work [14]. If an attacker can extract the initial random symmetric key, then it is possible for the attacker to compute all past and future keys. Therefore, a high quality random number generator is very important for the effectiveness of symmetric key operations. secFleck provides a `fos_tpm_rand()` primitive, which is based on a true Random Number Generator (RNG), and is Federal Information Processing Standards (FIPS) 140-2 compliant.

Each TPM has a unique 2048-bit private key established during manufacture which cannot be read. However an application can acquire the corresponding public key from the TPM which can be shared with other nodes for encryption and signature verification purposes. An application encrypts a message by providing the plain text, the length of the plain text, and a public key — the cipher text is returned. Similarly, an application can decrypt cipher text. The secFleck encryption and decryption facilitates message confidentiality.

secFleck provides two additional primitives, i.e., `fos_tpm_sign()` and `fos_tpm_verifySign()`, for applications to sign messages or to verify the signatures of messages. The `digest` parameter in these two primitives are generated by the Secure Hash Algorithm (SHA-1) of plain messages. FOS also provides a function for computing SHA-1.

A base station typically has more computation, memory and energy resources and can be treated as a Certificate Authority (CA). All the nodes store the CA's public key in their permanent memories such as EEPROM before deployment, and the base station has the public keys of all nodes. Multiple base stations and/or dedicated CA nodes with more memory can be used to improve the scalability of this approach. Therefore, message authenticity can be facilitated.

```
1  /* Duty cycle TPM chip functions. */
2  uint8_t fos_tpm_startup(void);
3  uint8_t fos_tpm_turnoff(void);
4
5  /* True random number generator. */
6  uint8_t fos_tpm_rand(uint8_t *randNumber, uint8_t len);
7
8  /* secFleck public key collector. */
9  uint8_t fos_tpm_getPubKey(uint8_t *pubKey);
10
11 /* Asymmetric key encryption/decryption. */
12 uint8_t fos_tpm_encryption(uint8_t *msg, uint16_t len,
13                 uint8_t *pubKey, uint8_t *cipher);
14 uint8_t fos_tpm_decryption(uint8_t *cipher, uint8_t *msg,
15                 uint16_t *len);
16
17 /* Digital signature and verification. */
18 uint8_t fos_tpm_sign(uint8_t *digest, uint8_t *signature);
19 uint8_t fos_tpm_verifySign(uint8_t *signature, uint8_t *pubKey,
20               uint16_t *digest);
21
22 /* Symmetric session key encryption/decryption. */
23 uint8_t fos_xtea_encipher(uint8_t *msg, uint8_t *key,
24                 uint8_t *cipher, uint8_t nRounds);
25 uint8_t fos_xtea_decipher(uint8_t *cipher, uint8_t *key,
26                 uint8_t *msg, uint8_t nRounds);
```

Fig. 4. secFleck application interface for public key infrastructure and symmetric session key functions. The public key cryptographic primitive interfaces allow applications to start up and turn off on-board TPM chip, to read the public key of the TPM chip, to encrypt or to decrypt a message, to sign a message, and to verify a signature. The symmetric key primitive interfaces allow applications to encrypt and decrypt messages by XTEA algorithm.

Symmetric Key (XTEA) FOS Functions. Previous work [14] show that symmetric key cryptography is tractable on mote-class hardware and can achieve message confidentiality. Further, symmetric key cryptography is significantly less resource-intensive than asymmetric key cryptography such as RSA. secFleck also features a 128-bit symmetric block cipher based on eXtended Tiny Encryption Algorithm (XTEA) [16]. XTEA operates on 64-bit blocks with 32 or 64 rounds. secFleck chooses XTEA symmetric key cryptography because of its small Random Access Memory (RAM) footprint, which makes it a good candidate for tiny sensor devices that typically have less than 10 KB RAM. XTEA can be used in an output feedback mode to encrypt or decrypt variable length strings.

4 Performance Evaluation

In this section, we discuss the performance of the secFleck platform in terms of computation time, energy consumption, and financial cost.

4.1 Asymmetric Key (RSA) Operations

As part of our benchmarking we also implemented the RSA encryption algorithm in software for comparison. Table 1 shows the encryption time for different key sizes and RSA public exponents (e) in both software and hardware implementation. The results show that the TPM chip can reduce the computation time of RSA encryption by a factor of $8,000$, when $e = 65{,}537$ and key size is 2048 bits. Table 1 also shows that software RSA implementation is impractical using embedded micro controllers such as Atmega 128 when $e > 3$ and for key size larger than 1024 bits. A small e will make RSA less secure, and a key size of 1024 bits will no longer be considered secure in a few years time.

We have not implemented the RSA decryption algorithm in software because it is significantly more computationally intensive than the RSA encryption algorithm (see Table 2). Table 2 also shows RSA encryption, decryption, sign and signature verification computation time in secFleck.

Table 3 shows the current consumption for different secFleck operations. It shows that RSA operations consume 37% to 65% more current than transmitting in secFleck. Table 4 shows the energy consumption of 2048-bit RSA encryption operation when $e = 65,537$. It shows that the software-based approach consumes around 1,300 times more energy compared to secFleck for an RSA encryption operation. Table 4 also shows that the software-based approach RSA encryption

Table 1. Comparison of RSA encryption times

Public Exponent (e)	Software 1024 bit	Software 2048 bit	Hardware 2048 bit
3	0.45s	65s	N/A
65,537	4.185s	450s	0.055s

Table 2. RSA computation time in secFleck for $e = 65,537$ and 2048 bit key

Encryption	Decryption	Sign	Verification
55ms	750ms	787ms	59ms

Table 3. secFleck current consumption

Module	Current (mA)
Fleck3 (without radio, node idle)	8.0
Fleck3 + Receive	18.4
Fleck3 + Transmit	36.8
Fleck3 + TPM encryption	50.4
Fleck3 + TPM decryption	60.8
Fleck3 + TPM signature	60.8
Fleck3 + TPM signature verification	50.4

in WSN is indeed impractical in terms of both computation time and energy consumption for reasonable RSA exponent and key size. On the other hand, secFleck makes it feasible to support PK technology for WSN.

4.2 Symmetric Key (XTEA) Operations

We tested the performance of XTEA cryptography to determine its computation speed on the Fleck platform. secFleck can encrypt one block of 64-bit data in approximately 1.15 ms. Therefore, it takes approximately 18 μs (Table 4, Row 3) to encrypt one bit data. Furthermore, the effective data rate of Fleck transceiver (Nordic nRF905) is 50 kb/s with Manchester encoding. For a 32-byte physical layer payload, there are a four-byte address and a two-byte Cyclic Redundancy Check-16 (CRC-16) overheads. Therefore, the available bandwidth for Media Access Layer (MAC) is $50 \times 32 \div (32 + 4 + 2) = 42.11$ kb/s. It takes 23.75 μs for a NRF905 transceiver to transmit one bit, which is significantly longer than the encryption time (17.97 μs).

Table 4 also shows that software symmetric key cryptography is indeed significantly faster than hardware RSA asymmetric key cryptography (18 μs vs. 27 μs per bit). Furthermore, XTEA encryption consumes approximately ten times less energy compared to hardware RSA encryption, and approximately 12,000

Table 4. secFleck (RSA and XTEA) encryption energy consumption for *one bit* of data

Platform	Current (mA)	Time (μs)	Energy (μJ)
RSA (software, $e = 65,537$, 2048 bit key)	8.0	219,730	7,030.0
RSA (hardware, $e = 65,537$, 2048 bit key)	50.4	27	5.4
XTEA (software, 128 bit key)	8.0	18	0.6

times less energy compared to software RSA encryption. It suggests that, in energy-impoverished WSN, we should use symmetric cryptography for most secure communications, and should use asymmetric cryptography in critical tasks only (i.e., the symmetric key management).

The other key advantage of XTEA is *space efficiency*. The FOS XTEA implementation has less than 100 lines of C codes, and requires 52 bytes of RAM and 1,082 bytes of program space only.

4.3 The Financial Cost of secFleck

An Atmel AT97SC3203S TPM chip costs $4.5 when ordered in quantities[1], which is less than 5% of the cost of popular sensor devices such as Telosb, Iris mote, and Fleck (about $100). The TPM chip is small in size (Figure 1) measuring just 6.1 × 9.7 mm and is less than 2% of the area of the Fleck and could be integrated onto a future version rather than the cumbersome expansion board used in this prototype.

5 Case Studies

In this section, we demonstrate the power of our secFleck primitives (shown in Fig. 4) to easily and efficiently realize secure WSN applications. These applications include, but are not limited to, secure over-the-air programming, secure Remote Procedure Calls (RPC), and secure session key management. We have chosen to implement variants of state-of-the-art key management [17] and secure software update protocol [13] with secFleck primitives, and show how secFleck primitives can improve the protocol's performance.

5.1 Symmetric Session Key Encryption/Decryption

Symmetric key cryptography consumes significantly less energy than RSA asymmetric key cryptography (see Table 4), as we envision that symmetric session key cryptography will be used for most WSN secure communications, and asymmetric cryptography will be used for limited critical tasks. For example, asymmetric cryptography is used to exchange a new symmetric key daily or hourly (also called the rekey process). We will discuss the rekey process in detail later.

By utilizing two secFleck primitives, it is easy to achieve symmetric key cryptography in secFleck (see Fig. 5). `fos_xtea_getkey()` (Line 4) reads a symmetric key from secFleck memory, and `fos_xtea_encipher()` encrypts a plain message (msg), and returns an encrypted message (cipher). Therefore, link-level secure transmissions can be achieved by passing the returned cipher over the radio.

An application can choose to store the session keys in Fleck RAM, EEPROM, or the TPM EEPROM. When the keys are stored in the Fleck RAM or

[1] http://www.atmel.com/dyn/products/view_detail.asp?ref=&FileName=embedded10_18.html&Family_id=620 (accessed on 18th June, 2008).

```
#DEFINE NROUNDS 64

/* XTEA encryption in secFleck. */
fos_xtea_getkey(key, location);
fos_xtea_storekey(key, location);
fos_xtea_encipher(msg, key, cipher, NROUNDS);
```

Fig. 5. XTEA encryption with secFleck primitives

EEPROM, the key (getting and storing) operations consumes significantly less energy than when the keys are stored in the TPM EEPROM. However, storing the keys in the Fleck also exposes the keys to more risks. Hartung et al. demonstrated how to extract the information in a node's EEPROM and RAM within one minute in [10]. Perhaps it is better to store the key in the TPM chip for those infrequent operations (e.g., sending one temperature sample to the base station every five minutes); store the key in the Fleck memory for those high-throughput operations (e.g., secure over-the-air-programming).

5.2 Sensor Node Symmetric Session Key Request/Assignment Operation

Fig. 6 shows the protocol for a sensor node (Node A) to request a new symmetric key from a base station. Node A initiates this process periodically, e.g., hourly or daily, by generating a random number (N_a) and encrypting N_a with the Request (Req) command using Base's public key (Pk_{Base}) before transmitting it to the base. After receiving the Request message from Node A, the base decrypts the message with its private key (SK_{Base}). The base then responds to the Req command by generating a new symmetric session key (K_{BA}), and encrypts it together with N_a using Node A's public key (Pk_A) before transmitting it to Node A. Node A decrypts the message from the Base with its private key (SK_A) and obtains the new symmetric key (K_{BA}). Node A and the base can then use K_{BA} for future secure communications. Fig. 6 also shows the five secFleck primitives associated with each step of the key request protocol.

The session key *assignment* operation is symmetric to the key *request* operations. The key assignment protocol is initiated, e.g., in an group key establishment event (see Section 5.3), by the base station instead of a node.

5.3 Group Key Establishment Operation

Group key establishment can be achieved by a combination of sensor node symmetric session key request operations and sensor node symmetric session key assignment operations. For example, if node A wants to communicate with node B and C, Node A will request a new group session key from the base station via the session key request operation introduced in Section 5.2. After receiving the key request operation from Node A, the base station generates a new symmetric

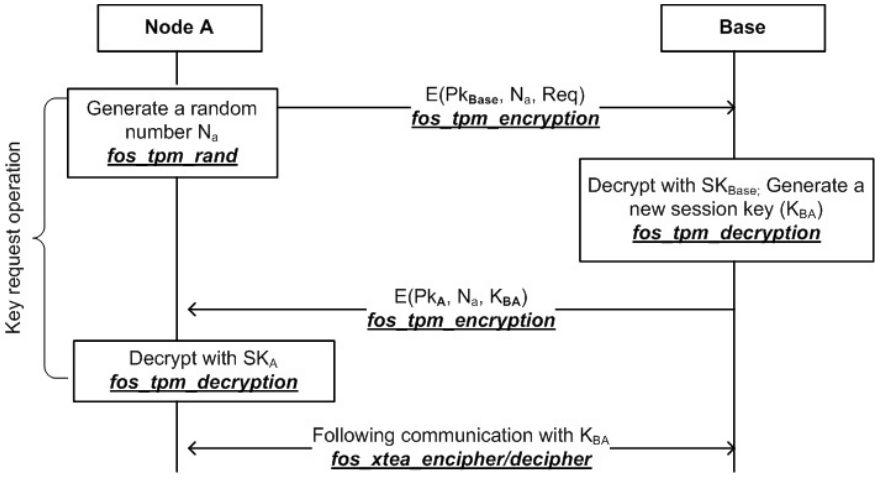


Fig. 6. Symmetric session key request operation with secFleck primitives (underline). Node A request a session key from a base station.

key (K_{abc}). The base station assigns K_{abc} to Node B and C via two session key assignment operations (see Section 5.2) before transmitting K_{abc} to Node A. Then, Node A, B, and C begin secure communications using group session key K_{abc}.

5.4 Secure Software Update Protocol

Multihop Over the Air Programming (MOAP) protocols such as Deluge [13] enable users to reprogram/retask the WSN remotely, which is critical to efficient and effective management of large-scale long-duration WSNs. The basic Deluge protocol works as follows. A node (Node A) advertises its program version periodically. One of its neighbors (Node B) will send a request to download a copy of the program from Node A if Node B's version is older than Node A's. Node A begins the download process after receiving the request. To support concurrent data disseminations and reduce network reprograming time, Deluge divides a program into a number of pages.

By using the group key establishment operation introduced in Section 5.3, secFleck can provide data confidentiality to Deluge. Furthermore, a base station can achieve integrity and authentication by signing the advertisement message and the program pages of Deluge with its private key (SK_{Base}) before disseminating it to the network. After receiving a program page or an advertisement message, a secFleck node can then verify the page or the advertisement message with the public key of the base station (Pk_{Base}). This mechanism ensures that wireless bootstrap can only be initiated by an authorized host, that the code stream is private, and that a page is not committed to flash unless it is from an authorized host.

A secFleck node can verify the signature of the a 256 byte page in 59 ms (see Table 2), which is more than 4,300 bytes/second. This secFleck signature verification rate is approximately 50 times faster than the average 88.4 bytes/second dissemination rate achieved by Deluge in a 75 node network [13].

5.5 Backward Secrecy and Forward Secrecy

secFleck can enhance the security levels of the rekey process by providing backward secrecy and forward secrecy. Backward secrecy means that compromising the current symmetric link key does not allow an attacker to learn previously recorded messages encrypted with the previous key. Forward secrecy means that compromising the symmetric link key does not allow an attacker to learn future communication encrypted with the following key.

A symmetric link key can be found by an attacker by extracting it directly from a captured node via a JTAG or similar device [10] because of the exposed nature of nodes in WSN. Furthermore, the attacker can also extract the initial random key used by the software pseudo-random number generator. This key allows the attacker to compute all past and future nonces used in the key updating protocol, which in turn allows the attacker to compute all past and future keys.

Equipped with a FIPS 140-2 compliant true Random Number Generator (RNG), secFleck can increase the security level of the protocols. It is very difficult, if not impossible, for the attacker, who has obtained the current symmetric link key, to find out the past or future keys generated by a true RNG. An application can obtain a true random number by calling `fos_tpm_rand()` primitive (Line 6, Fig. 4).

5.6 Secure Remote Procedure Calls

The Fleck Operating System (FOS) uses Remote Procedure Calls (RPC) to allow application programs to seamlessly access services on one or more sensor nodes.

Each node-side service is described by an action file, a C-like function that supports multiple input and output arguments. A code generator, in Python, parses all action files and generates a server function and all the serializing and deserializing code, as well as a Python class to be used by base station applications. All nodes support the common set of actions listed in Table 5, in addition to application specific actions.

An RPC call message comprises the function arguments, the function enumerator, sequence number, node id of the caller and a CRC-32. Except for the `assign_session_key` and `request_session_key` RPC messages, all the other RPC messages are encrypted using XTEA with the current session key (see Section 5.1). `assign_session_key` and `request_session_key` RPC messages are encrypted and signed with PKC introduced in Section 5.2. On receipt of an RPC call message (indicated by the routing header type) the message is decrypted using the session key and the CRC-32 checked.

Table 5. Common secure FOS RPC actions

RPC actions	Description	Cryptography
assign_session_key	assign a new symmetric session key to a node	PK
request_session_key	request a new symmetric session key from a base	PK
kernel	get FOS system memory statistics	share
read_eeprom	read from EEPROM	share
read_ram	read from RAM	share
threads	get information about threads, label and stack usage	share
write_eeprom	write to EEPROM	share
write_ram	write to RAM	share
rtc_get	get time from the real-time clock	share
rtc_set	set the real-time clock	share
txpwr_set	set radio transmit power	share
leds	set or toggle LEDs	share
power	get battery and solar cell status	share

In a sensor network, it is possible to broadcast the RPC call encrypted by a group symmetric key (see Section 5.3), and have the function executed in parallel on many nodes which all return their results to the caller. In this case the result of an RPC call would be a list of return values rather than just one.

Secure RPC, based in secFleck primitives, provides privacy of commands and return values, authentication and immunity to replay attacks.

6 Related Work

In this section, we provide a brief overview of secure communications most directly relevant to secFleck.

Rivest Shamir Adelman (RSA) is the most widely used Public Key Cryptography (PKC) in the Internet, and a comprehensive guide to RSA is available in [18]. RSA is much slower than Xtended Tiny Encryption Algorithm (XTEA) [16] and other (shared) symmetric cryptography such as TinySec [14].

It is our thesis that most of the secure communications in resource-constrained WSN will be based on symmetric cryptography. A symmetric key can be discovered by an attacker by extracting it directly from a captured node via a JTAG or similar device [10] because of the distributed and embedded nature of nodes in WSN. Therefore, an effective symmetric key establishment and management scheme is of prime importance. RSA and Diffie Hellman key agreement techniques [8] are widely used key agreement protocols in the Internet, but have been previously considered infeasible for WSNs because of the resource constraints of sensor devices.

Researchers have proposed a limited version of RSA PKC (TinyPK) that performs encryption operations only and uses smaller RSA parameters such

as public exponents and key sizes [22]. However, the security levels of RSA cryptography is severely compromised by using smaller public exponents and key sizes. Recently, the importance of symmetric key cryptography and the critical roles of key management mechanism in WSN was observed by Nilsson et al. [17] who proposed an efficient symmetric key management protocol for the WSN. However, they have focused on the protocol design and formal verification, and have not addressed the resource constraint problems in implementing the protocol.

The research community is developing faster and more energy efficient PKC algorithms such as Tiny Elliptic Curve Cryptography TinyECC [15] for the resource-impoverished WSN. While TinyECC shows the most promise to run at usable speeds on WSN nodes [9], there are concerns related to patents, which are one of the main factors limiting the widely acceptance of ECC. In this regard we note that the RSA and XTEA algorithms used in this work is in the public domain.

While Multihop Over the Air Programming (MOAP) protocols [13] enable application users to program and reprogram WSNs easily, it also opens the door for unauthorized users to implant malicious code into the WSN. Dutta et al. attempt to secure the MOAP [7] by introducing program authenticity and integrity with a cut-down version of software-based RSA PKC similar to TinyPK [22]. As in TinyPK, the security levels of RSA cryptography will be compromised by using smaller RSA public exponents and key sizes.

Believing that PKC such as RSA and ECC is too resource-intensive for the resource-impoverished WSN, researchers have investigated alternative methods to ensure program authenticity and integrity by secure hash chain, hash tree and/or their hybrid [5,20].

In contrast to the existing alternatives of PKC that typically have limited functions, secFleck provides E-Commerce level PKC, which facilitates secure communication services such as confidentiality, authenticity and integrity with low financial overhead, by exploiting the capability of a commodity Trusted Platform Module (TPM) chip.

7 Conclusion and Future Work

We have presented secFleck, a TPM-based PK platform for sensor networks that facilitates message security services such as confidentiality, authenticity and integrity. Our evaluation shows that secFleck provides Internet-level public-key services quickly, with low energy consumption and at low cost in terms of parts and board size. This is followed by examples, built on secFleck, of symmetric key management, secure RPC and secure software update, which demonstrates that the secFleck platform is easy-to-use.

Our next step is to investigate other features of TPM modules such as secure storages and remote attestations.

Acknowledgments

The authors thank Hailun Tan (Univeristy of New South Wales, Australia), Dr. Juanma Gonzalez Nieto (Queensland University of Technology, Australia) and the anonymous reviewers for their comments and suggestions.

References

1. Habitat monitoring on great duck island, http://www.greatduckisland.net/index.php
2. Habitat monitoring on james reserve, http://www.jamesreserve.edu/
3. Bernstein, B.: Circuits for integer factorization: A proposal (manuscript, 2001), http://cr.yp.to/papers.html
4. Corke, P., Sikka, P.: Demo abstract: FOS — a new operating system for sensor networks. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913. Springer, Heidelberg (2008)
5. Deng, J., Han, R., Mishra, S.: Secure code distribution in dynamically programmable wireless sensor networks. In: IPSN 2006: Proceedings of the fifth international conference on Information processing in sensor networks, pp. 292–300. ACM Press, New York (2006)
6. Dinh, T.L., Hu, W., Sikka, P., Corke, P., Overs, L., Brosnan, S.: Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In: Second IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2007), Dublin, Ireland (October 2007)
7. Dutta, P.K., Hui, J.W., Chu, D.C., Culler, D.E.: Securing the deluge network programming system. In: IPSN 2006: Proceedings of the fifth international conference on Information processing in sensor networks, pp. 326–333. ACM Press, New York (2006)
8. Goldwasser, S.: New directions in cryptography: twenty some years later (or cryptograpy and complexity theory: a match made in heaven). In: FOCS 1997: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS 1997), Washington, DC, USA, p. 314. IEEE Computer Society Press, Los Alamitos (1997)
9. Gurn, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing elliptic curve cryptography and rsa on 8-bit cpus. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
10. Hartung, C., Balasalle, J., Han, R.: Node compromise in sensor networks: The need for secure systems. Technical report, University of Colorado at Boulder (January 2005)
11. Hill, J., Culler, D.: Mica: a wireless platform for deeply embedded networks. IEEE Micro 22(6), 12–24 (2002)
12. Hu, W., Tran, V.N., Bulusu, N., Chou, C.T., Jha, S., Taylor, A.: The design and evaluation of a hybrid sensor network for cane-toad monitoring. In: IPSN 2005: Proceedings of the 4th international symposium on Information processing in sensor networks, Piscataway, NJ, USA. IEEE Press, Los Alamitos (2005)
13. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: SenSys 2004: Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 81–94. ACM Press, New York (2004)

14. Karlof, C., Sastry, N., Wagner, D.: Tinysec: a link layer security architecture for wireless sensor networks. In: SenSys 2004: Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 162–175. ACM Press, New York (2004)
15. Liu, A., Ning, P.: Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In: IPSN 2008: Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008), Washington, DC, USA, pp. 245–256. IEEE Computer Society Press, Los Alamitos (2008)
16. Needham, R., Wheeler, D.: Tea extensions. Technical report, University of Cambridge (October 1997)
17. Nilsson, D.K., Roosta, T., Lindqvist, U., Valdes, A.: Key management and secure software updates in wireless process control environments. In: WiSec 2008: Proceedings of the first ACM conference on Wireless network security, pp. 100–108. ACM Press, New York (2008)
18. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
19. Shamir, A., Tromer, E.: On the cost of factoring rsa-1024. RSA CrytoBytes 6(2) (2003)
20. Tan, H., Jha, S., Ostry, D., Zic, J., Sivaraman, V.: Secure multi-hop network programming with multiple one-way key chains. In: WiSec 2008: Proceedings of the first ACM conference on Wireless network security, pp. 183–193. ACM Press, New York (2008)
21. Wark, T., Crossman, C., Hu, W., Guo, Y., Valencia, P., Sikka, P., Corke, P., Lee, C., Henshall, J., Prayaga, K., O'Grady, J., Reed, M., Fisher, A.: The design and evaluation of a mobile sensor/actuator network for autonomous animal control. In: IPSN 2007: Proceedings of the 6th international conference on Information processing in sensor networks, pp. 206–215. ACM Press, New York (2007)
22. Watro, R., Kong, D., Cuti, S.-f., Gardiner, C., Lynn, C., Kruus, P.: Tinypk: securing sensor networks with public key technology. In: SASN 2004: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, pp. 59–64. ACM Press, New York (2004)

Accurate Network-Scale Power Profiling for Sensor Network Simulators

Joakim Eriksson, Fredrik Österlind, Niclas Finne,
Adam Dunkels, Nicolas Tsiftes, and Thiemo Voigt

Swedish Institute of Computer Science
{joakime,fros,nfi,adam,nvt,thiemo}@sics.se

Abstract. Power consumption is the most important metric in wireless sensor network research, but existing simulation tools for measuring or estimating power consumption are either impractical or have unclear accuracy. We present COOJA/MSPSim, a practical simulation-based tool for network-scale power estimation based on Contiki's built-in power profiling mechanism, the COOJA sensor network simulator and the MSPSim sensor node emulator. We compare experimental results measured on real sensor nodes with simulation results for three different MAC protocols. The accuracy of our results indicates that COOJA/MSPSim enables accurate network-scale simulation of the power consumption of sensor networks.

1 Introduction

Power consumption is the most important metric in wireless sensor networks because reduced power consumption leads to increased network lifetime. Many different mechanisms for reducing the power consumption for sensor networks have been proposed. Energy has been reduced with more efficient topology management [4], routing [20] and medium access [1,24]. Most power-saving mechanisms focus on reducing radio on-time because radio communication and idle listening are the most power-consuming task in wireless sensor networks [8,18]. To evaluate the efficiency of power-saving mechanisms, researchers must be able to quantify the energy consumption at the network scale.

Software-based power profiling has enabled non-intrusive and scalable power profiling in real sensor networks [7]. The technique is based on measuring the time that each component is active and multiplying that time by the component's power consumption. This method of measuring energy is accurate, but by the nature of testbeds, it is typically limited in scale and mobility. Testbed experiments require setup, instrumentation and infrastructure. Furthermore, the arrangement of the nodes is usually fixed and difficult to change. Simulations, on the other hand, scale well and handle mobility and repeatability with ease. There exist a number of simulators for sensor networks. Some of them are able to estimate power consumption but their accuracy has only been demonstrated in node-local experiments.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 312–326, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In this paper we present COOJA/MSPSim, a power profiling tool that enables accurate network-scale energy measurements in a simulated environment. Our tool combines the sensor network simulator COOJA [16] with the MSPSim hardware emulator [9] and Contiki's software-based power profiler [7]. By using the detailed instruction level emulation of the MSP430 processor, we can obtain accurate power profiles of simulated networks in COOJA/MSPSim.

The contributions of this paper are the presentation and the evaluation of COOJA/MSPSim. Our results demonstrate that COOJA/MSPSim enables accurate network-scale power profiling for sensor networks. Our evaluation consists of three case studies. In each case study we use a different MAC protocol to explore power profiling nuances as far down as possible in the network stack. The MAC protocols are Low Power Probing [15], X-MAC [1], and a TDMA-based data collection protocol called CoReDac [25]. Our case studies demonstrate that the power measurements for both transmission and listen power in testbed and simulation match very well. CoReDac's transmission power is simulated with an accuracy of around $0.6\mu W$. Using LPP, we simulate the power consumption in high packet transmission rate scenarios that matches the testbed results with a difference of less than 0.8%. For the more complicated X-MAC protocol, the difference between the experimental and simulated results is typically below 2%.

The remainder of this paper is outlined as follows. We explore related work in Section 2. Then we describe our tool for network-scale power profiling in Section 3. In Section 4 we experimentally evaluate the accuracy of COOJA/MSPSim through a set of case studies with different MAC protocols. Section 5 concludes our work.

2 Related Work

There are many sensor network simulators with energy estimation abilities [12, 13, 21], but their accuracy has only been demonstrated in node-local experiments. Avrora [22] is a machine code level simulator similar to MSPSim. While it offers a cycle-accurate simulation of AVR-based nodes, it does not have a power profiler. For this purpose, Landsiedel et al. have created the power analyzer AEON [12] on top of Avrora. AEON is limited to TinyOS, however. Furthermore, the authors do not compare simulation with testbed results for multi-hop applications. In contrast, our work is aimed at power profiling at a network scale.

PowerTOSSIM [21] is an extension of TOSSIM [13] designed to estimate the power consumption of Mica2 sensor nodes. Since TOSSIM operates at the operating system level, its granularity with respect to timing and interrupt properties is not sufficient when nodes interact [22]. Our measurements of the radio power consumption show that a very detailed model of the node is required to obtain accurate results. Trathnigg et al. [23] improve the accuracy of PowerTOSSIM, but for a single node only. Colesanti et al. [5] evaluated the accuracy of a multi-node simulation using metrics such as packets sent and received. They got inaccurate results and concluded that a more sophisticated node model is required. By using emulated nodes in the simulation, COOJA/MSPSim uses a very detailed node model that considerably improves the power measurement accuracy.

Haq and Kunz compare emulated testbed results with simulation results for mobile ad-hoc network routing protocols [10]. While their results match in low traffic scenarios, the results differ in scenarios with higher traffic rates. In contrast, COOJA/MSPSim maintains the accuracy in high traffic. Cavin et al. compare simulation results obtained with several simulators for mobile ad-hoc networks [3]. They discovered large divergences for packet delivery rates, latency and message overhead when simulating an intentionally simple flooding algorithm in different scenarios.

Ivanov et al. show that after careful simulation parameter adjustment, NS-2 accurately models packet delivery ratios in a wireless mesh testbed [11]. The parameter adjustment did not improve the accuracy regarding connectivity and packet latencies, however. COOJA/MSPSim allows accurate power profiling of arbitrary nodes in the network, and is orthogonal to that of an accurate radio model.

3 Simulation-Based Network-Scale Power Profiling

We develop COOJA/MSPSim for network-scale power estimation by combining three existing tools: Contiki's power profiling [7], the COOJA sensor network simulator [16], and the MSPSim sensor node emulator [9].

3.1 Contiki Power Profiler

The built-in power profiler in Contiki estimates the power consumption of sensor nodes in a real network. Thereby it enables scalable measurements of the power consumption. The power profiling mechanism measures the time that hardware components spend in different operating modes. This data is then combined with detailed pre-measured data of power consumption for these components into power consumption estimations. This mechanism can be implemented on most microprocessors with very small overhead.

3.2 COOJA

The other important component of our power profiling software is the COOJA simulator, a Java-based sensor network simulator. COOJA has the ability to mix simulations of sensor devices at multiple abstraction levels. These levels are application level, OS level, and hardware level. In the application level the simulated nodes run the application logic reimplemented in Java - the native language of COOJA. In the OS level the nodes use the same code as real nodes, but compiled for the host machine running COOJA. Finally in the hardware level the nodes run the same compiled code that can be used in real nodes, e.g. the same system image. The hardware level is provided by MSPSim that emulates systems based on the MSP430 processor family. By using MSPSim underneath, COOJA allows simulated nodes to execute the same system image as the one used on the real nodes. The nodes at different abstraction levels communicate with each other using one of the three radio propagation models available in COOJA.

3.3 MSPSim

MSPSim is an instruction level emulator of MSP430-based sensor network nodes. MSPSim targets cycle accurate emulation of both the MSP430 CPU core and built-in peripherals such as timers, serial communication and analog to digital converters. Furthermore, MSPSim emulates external components such as the radio chip CC2420, sensors, and flash memories. MSPSim also provides emulation of complete sensor devices such as the Tmote Sky [17] and Scatterweb ESB [19].

3.4 A Network-Scale Power Profiler

We combine the three tools presented above into an accurate network-scale power profiler. Figure 1 shows the integrated COOJA/MSPSim architecture with COOJA controlling MSPSim and the power profiler in Contiki that provides COOJA with the estimation of energy consumption. It also shows the sensor nodes' radio communication via the emulated CC2420 and COOJA's radio medium simulation. Several improvements of the involved tools are necessary to achieve an accurate power profiling. We ensure that the components emulated in MSPSim have the same timing as in real nodes. In addition, we integrate MSPSim more tightly with COOJA including a more fine-grained connection between the emulated nodes' radio chips. In order to increase the power profiling accuracy we extend the timer emulation, and improve the timing precision of the SPI bus and the CC2420 radio transmissions.

We estimate the power consumption on a network scale in COOJA using Contiki's built-in power profiling mechanism, and we run the Contiki application on

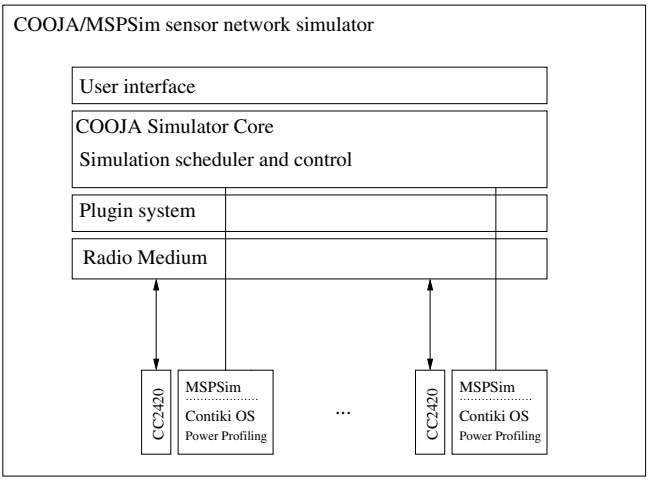


Fig. 1. The architecture of the COOJA/MSPSim simulator. MSPSim is integrated into the COOJA simulator and Contiki's built-in power profiler provides estimation of power consumption.

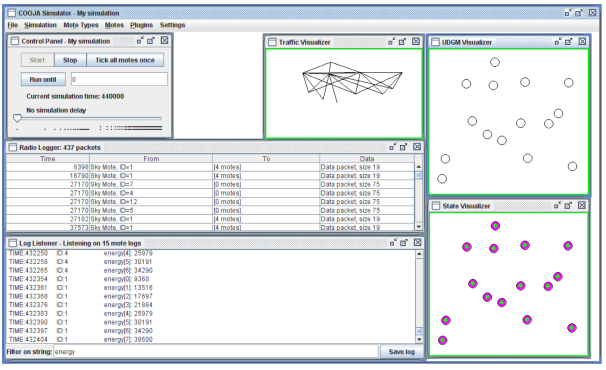


Fig. 2. Screenshot of power profiling in the COOJA/MSPSim simulator

emulated hardware using MSPSim. In this paper our simulations make use of hardware-emulation for all nodes since we need accurate power profiling for the complete network. Hence, we benefit both from COOJA's ability to simulate network behaviour and the on-line energy estimation in Contiki. Furthermore, MSPSim's sophisticated and detailed node model provides fine-grained timing and interrupt modeling which is important for the accuracy when estimating power consumption. Figure 2 shows a screenshot of our COOJA/MSPSim simulator during one of the evaluation experiments.

4 Evaluation

To evaluate the accuracy of our simulation-based approach, we compare the results of the energy estimation obtained through simulation with results obtained through testbed experiments. For the testbed experiments, we implement all software in the Contiki operating system [6] and run it on Tmote Sky nodes. The nodes use Contiki's software-based method to measure power consumption as described in Section 3.1. We perform the first experiments with a tree-based data collection protocol CoReDac. Then we experiment with the MAC protocols Low Power Probing [15] and X-MAC [1] to evaluate our power profiling accuracy on the lowest levels of the network stack. To compute the power consumption, we assume a voltage of 3V and a current draw of 20mA for listening and 17.7mA for radio transmissions, as measured on Tmote Sky nodes by Dunkels et al. [7].

4.1 Case Study: Data Collection with CoReDac

Protocol Overview For this case study we use CoReDac, a TDMA-based convergecast protocol [25]. In contrast to other convergecast protocols such as Dozer [2], CoReDac builds a collection tree that guarantees collision-free radio traffic.

To achieve low delay, CoReDac borrows the idea of staggered communication from D-MAC [14] as shown in Figure 3. In D-MAC packets from nodes on the same level can cause collisions, whereas CoReDac parent nodes avoid collisions

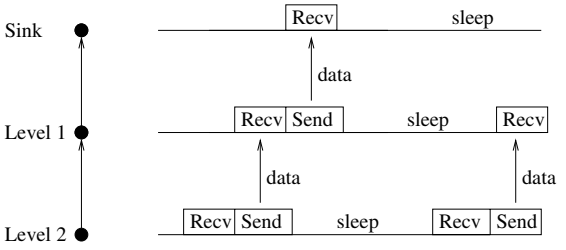


Fig. 3. Staggered communication in CoReDac

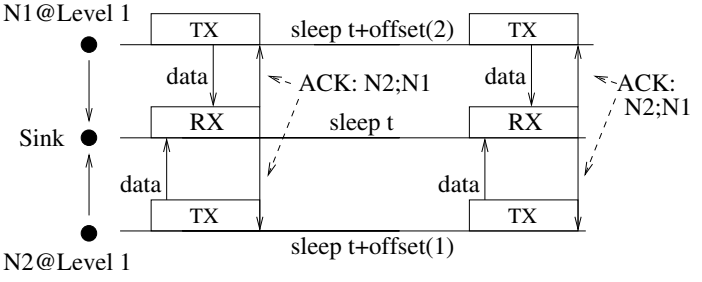


Fig. 4. On-demand slot assignment to avoid collisions

among packets from their children by assigning time slots for transmission to their children. The information about the assignment is contained in the acknowledgements. Acknowledgements play a pivotal role since they are also used for synchronization and on-demand slot assignment.

Figure 4 shows how CoReDac assigns transmit slots. The figure shows that the sink announces that $N2$ receives the transmit slot before $N1$. The sink's acknowledgement also signals when the sink's next receive slot starts, namely in $sleep_t$ seconds. This way, the acknowledgements contain all information to achieve a collision-free communication schedule between a parent node and its children. This scheme is recursively applied towards the whole tree. In order to avoid collisions between nodes on different levels, we set a maximum number of children per node. Based on this maximum number, its position in the tree and the receive slot of its parent, a node can compute its unique receive slot.

Setup and Results. We measure CoReDac's energy-efficiency both on real hardware with Contiki's built-in power profiling mechanism and with COOJA/MSPSim as described in Section 3. In these experiments, the maximum number of children is set to three and $sleep_t$ is set to 30 seconds. We compare networks of different sizes namely with 4 and 15 nodes. We also simulate a network of 40 nodes. Due to the limited size of our testbed, we can simulate a network consisting of only 15 nodes. We are able to simulate more than 40 nodes, but then we need to increase $sleep_t$ to guarantee collision-free trees. The length of the receive slot is not dependent of the number of nodes in a network. In our

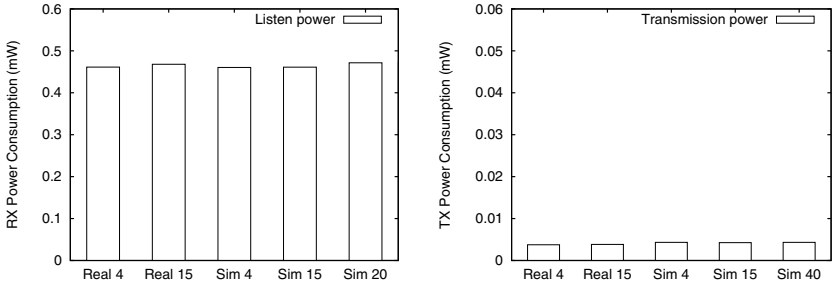


Fig. 5. The results from a testbed measurement of the power consumption of CoReDac (Real) and the simulation runs (Sim) agree with each other. The left graph shows the power consumption of the radio in listen mode and the right graph the power consumption of the radio in transmission mode. Note that the scales are different.

CoReDac implementation, there is a small difference between the length of the slots of the children of the same parent that depends on the order of the children. Therefore, we expect that the power consumption is independent of the size of the network but not exactly constant.

The left graph in Figure 5 shows CoReDac's average power consumption per node of the radio in listen mode that we call RX power consumption. Real n denotes results from a testbed measurement with n nodes, whereas Sim n denotes simulation results with n nodes. The figure shows that the measured power consumption on real nodes matches very well with the power consumption estimated with COOJA/MSPSim. In particular, the difference between the results does not increase with the size of the network.

The right graph in Figure 5 presents the average power consumption per node for transmissions.The figure shows that the power consumption for transmitting packets in CoReDac is less than 1% of the power consumption for listening and receiving packets which confirms the measurements by Dunkels et al. [7]. As for power consumption of the radio in listen mode, the results obtained by simulation and experiments with real hardware match well. The difference of the power consumption for transmitting packets is less than $0.6\mu W$. Further, the difference between the results does not increase with the size of the simulated networks. These results show that COOJA/MSPSim accurately power profiles networks of nodes running TDMA-based MAC protocols.

4.2 Case Study: Low Power Probing

Protocol Overview. Low power probing (LPP) is a power-saving MAC protocol [15]. As shown in Figure 6, LPP receivers periodically send small packets, so called probes, to announce that they are awake and ready to receive a data packet. After sending a probe, the receiver keeps its radio on for a short time to listen for data packets. A sender that has a packet to be sent turns on its radio waiting for a probe

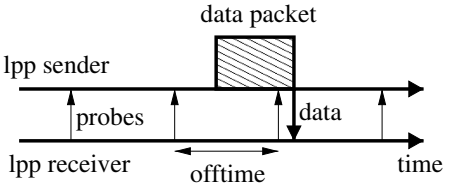


Fig. 6. Low Power Probing

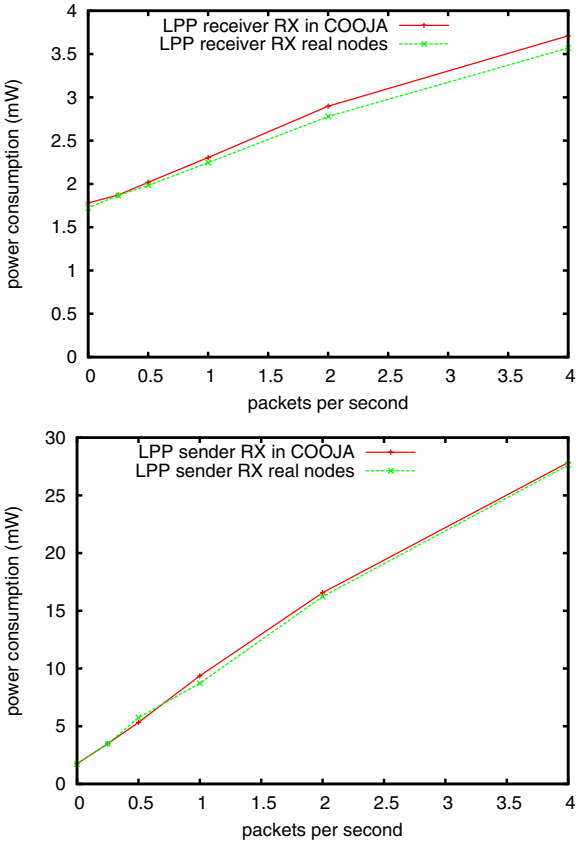


Fig. 7. With LPP as the underlying MAC protocol the results of the RX power consumption of the receiver (top graph) and the sender (bottom graph) is accurately simulated

from a neighbour it wants to send to. On the reception of a probe from a potential receiver, the node sends an acknowledgement before the data packet.

The LPP implementation in Contiki contains two important parameters. *On-time* determines how long a receiver keeps the radio on after the transmission of a probe. *Offtime* is the time between probes. Nodes modify *offtime* with ran-

dom offsets in order to avoid synchronization effects. The random offsets are distributed uniformly between $\frac{3}{4} \times offtime$ and $\frac{5}{4} \times offtime$.

Results: Sender-Receiver Scenario. In the first scenario we have one LPP receiver and one LPP sender. We vary the packet rate at which the sender hands data packets to the MAC layer. In the experiments, we set *ontime* to $\frac{1}{128}$ seconds and *offtime* to 0.25 seconds.

The results of this experiment are shown in Figure 7 and Figure 8. The top graph of Figure 7 depicts the RX power consumption for the receiver when the transmission rate of the sender increases from zero packets to four packets per second. The figure shows that the basic power consumption of an LPP receiver is about 1.75 mW, namely 1.78 mW in the simulator and 1.725 mW on real nodes. Note that this is very close to the theoretical value that is $\frac{1}{33} \times 20mA \times 3V = 1.818mW$ with the assumptions above. The power is consumed for keeping the radio on after the transmission of the probes. The power consumption increases when more packets need to be received since the packet reception requires the radio to be turned on longer than *ontime*. The figure also shows that the estimated power consumption in the simulator matches the power consumption measured with real hardware.

With a packet rate of four packets/s, the sending rate is higher than the probing rate and hence packets need to be dropped. Nevertheless, both the energy consumption and the packet reception are accurately simulated. In the simulation the packet rate is 87.9% on real nodes while it is 89.9% in simulation. The energy consumption with 4 packet/s is very accurate with a difference of less less than 4% for the receiver and less than 0.8% for the sender. The results clearly demonstrate that we do not encounter the problems that Haq and Kunz observed when comparing simulation and emulation of mobile ad hoc routing protocols, namely a large quantitative and qualitative difference under high traffic load [10]. The difference for lower packet rates is much smaller: for a packet rate of 0.25 packets/s less than 0.3% for the receiver.

The bottom graph of Figure 7 depicts the RX power consumption of the sender. Again, the results obtained by simulation match the results obtained with real hardware very well. The figure also shows that as expected the power consumption of the sender increases with a higher packet sending rate. With a higher sending rate, the overall time a sender has its radio on waiting for probes increases which causes higher RX power consumption.

Figure 8 shows that also the TX power consumption of both sender and receiver are accurately simulated despite that the TX power consumption is very low and hence only small timing differences could cause large relative discrepancies.

Results: Multi-hop scenario. In the next experiment, we place a sender, a forwarder and the sink in radio range of each other. All nodes run LPP with the same configuration parameters as above. The sending rate of the sender is set two packets/s. During our experiments, we do not experience any packet drops, i.e. all packets arrive at the sink.

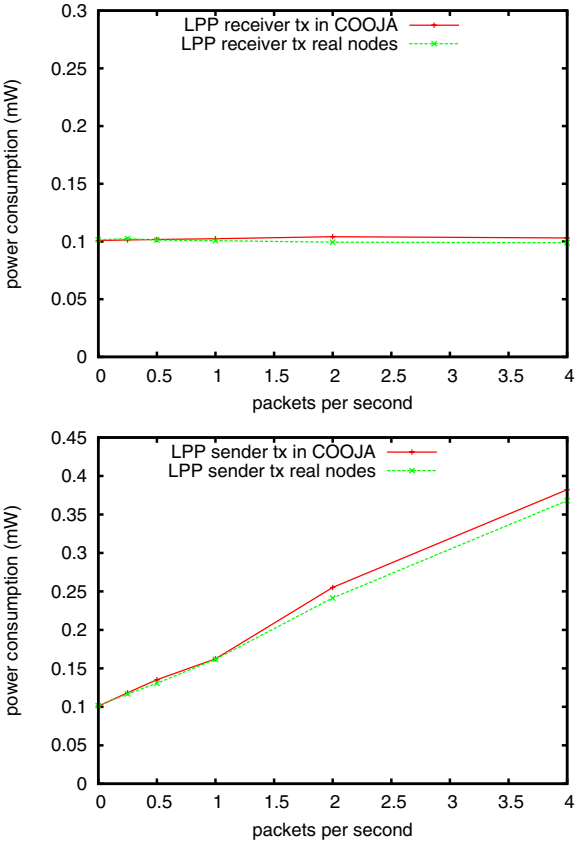


Fig. 8. With LPP as the underlying MAC protocol also the TX power consumption of the receiver (top graph) and the sender (bottom graph) are accurately simulated

Figure 9 shows the difference of the RX power consumption between the simulation and the experiments on real nodes. For the sink, the difference is below 2%. For the other two nodes, the difference is higher. When transmitting a packet, these nodes need to keep the radio on until they receive a probe. Probes are not sent at constant intervals to avoid synchronization effects, which is one possible reason for the larger difference between simulation and results with real nodes for nodes that transmit packets.

4.3 Case Study: X-MAC

Protocol Overview. X-MAC is a power-saving MAC protocol [1] in which senders use a sequence of short preambles (strobes) to wake up receivers. Nodes turn off the radio for most of the time to reduce idle radio listening. They wake up shortly at regular intervals to listen for strobes. When a receiving node wakes up and receives a strobe with its receiver address, the receiver replies with an acknowledgement

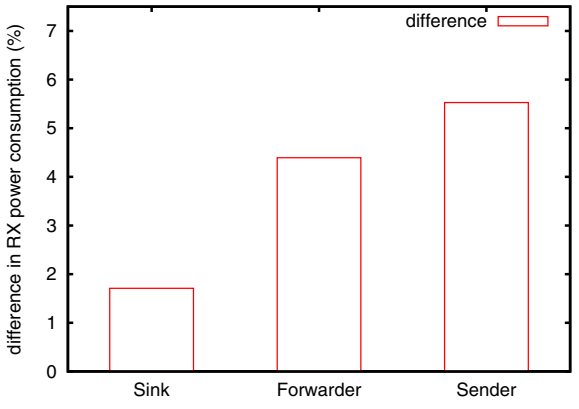


Fig. 9. The difference between simulated and measured RX power consumption for a two-hop network is small

indicating that it is awake. On the reception of the acknowledgement, the sender transmits the full packet.

The X-MAC implementation in the Contiki operating system contains two important parameters, namely *ontime* that determines how long a receiver keeps the radio on when listening for strobes, and *offtime*, the time between two listening times.

During the tests *ontime* is set to $\frac{1}{100}s$ and *offtime* is set to $\frac{1}{4}s$. During the evaluation experiments the sending application sends data at a fixed packet rate but with small variations in order to avoid synchronization between the sender and receiver that would lead to a constant number of strobes before the receiver wakes up. By doing this the average number of strobes required before wake-up of the receiver is half the maximum strobes, which gives an average wake-up time of $offtime/2$.

Results: X-MAC Sender and Receiver. The set-up of our first experiment with X-MAC consists of two nodes, one sender and one receiver. We use several different

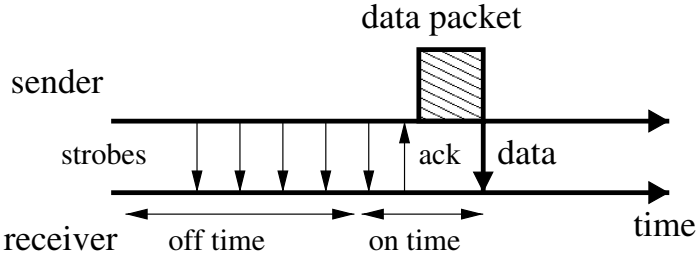


Fig. 10. X-MAC: the sender strobes until the receiver is awake and can receive a packet

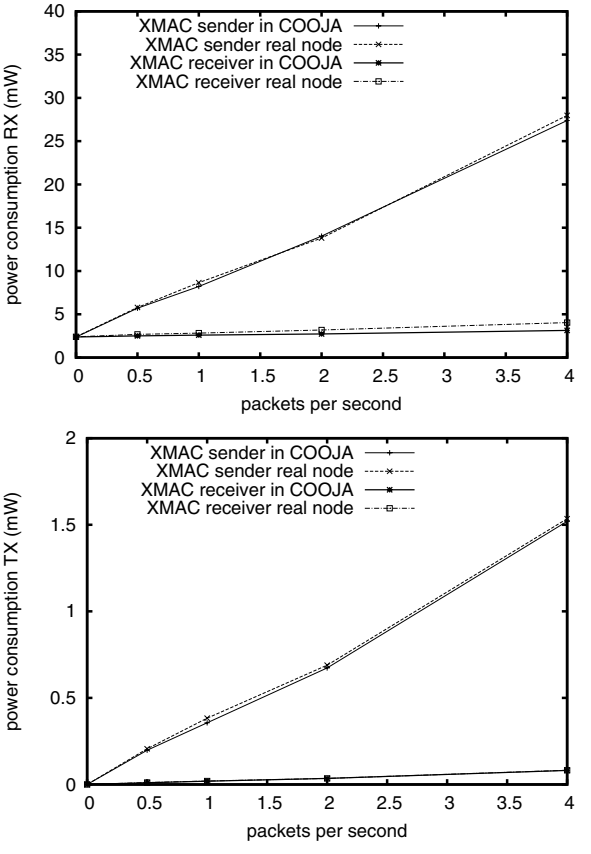


Fig. 11. X-MAC power consumption of radio listen (top graph) and radio transmissions (bottom graph). Both real nodes and simulated nodes have very similar behaviour for the senders and receivers when varying the packet ratio. This shows that the simulation has a good accuracy when simulating X-MAC.

packet rates ranging from one packet per two seconds to four packets a second. The latter is also the maximum speed with the used X-MAC configuration. Figure 11 shows the result of the measurements for both simulated and real nodes. The top graph shows the power consumption of the radio while being in receive mode, whereas the bottom graph shows the transmission power. The difference between the average energy consumption in simulated nodes and real nodes is small, typically around two percent. The result for a packet rate of four packets/s differ more. Initial experiments indicate that the main problem is the speed of communication between the microprocessor and the radio chip for reading out received packets.

During the initial runs on real sensor nodes we observed a packet loss below one percent. This packet loss was not simulated causing a small difference in packet loss between simulation and real nodes. Since the packet loss is below one percent this difference does not affect the results significantly.

The results depicted in Figure 11 from both the simulations and real nodes show that the average energy consumption for a X-MAC sender corresponds well with our expectation of sending strobes for half the *offtime* before waking the receiver.

4.4 Power Profiling Accuracy

Our experiments suggest that the results obtained through simulation match the results obtained through testbed experiments and Contiki's power profiling mechanism for different MAC protocols including TDMA-based protocols, low power probing and X-MAC low power listening. The results demonstrate that COOJA can accurately estimate the energy consumption of interacting nodes. For the TDMA-based CoReDac protocol we have shown that the simulation results also match results obtained in our testbed.

Our results also confirm the findings of Colesanti et al. that argued for better node models for improved accuracy [5]. Indeed, the results in Figure 8 initially differed with 50%, i.e. the power consumption was 0.1 mW on real nodes compared to 0.15 mW in simulation. Since transmitting a packet is inherently fast, even small inaccuracies can be responsible for the divergence of the results. By improving the accuracy of the timer system and the radio chip emulation in MSPSim, we were able to almost eliminate the discrepancy and achieve a difference of less than $0.6\mu W$ between simulation and testbed results.

As mentioned in Section 2, our goal is not to validate the accuracy of the radio models in COOJA but the radio power consumption caused by the interaction of nodes. The TDMA-based CoReDac protocol is collision-free by design and hence there are no collisions between nodes after the initialization phase. Therefore, we were able to validate CoReDac's simulated power consumption also in the testbed. For LPP and X-MAC the actual power consumption depends very much on the delivery rate, the risk of packet collisions and other factors that are influenced by the environment. Therefore, we have constrained ourselves to demonstrate COOJA's ability to accurately simulate behaviour that relies on fine-grained timing and interrupts when nodes interact. Nevertheless, our results indicate that when appropriately modeling the surroundings, COOJA enables accurate simulation of the power consumption of large-scale sensor networks.

5 Conclusions

In this paper we have presented COOJA/MSPSim, a tool for simulation-based network-scale power profiling that combines Contiki's on-line power profiling mechanism, the COOJA sensor network simulator and the MSPSim sensor node emulator. We have shown that the results obtained with COOJA/MSPSim correspond well with the results obtained through measurements on real hardware. Our results demonstrate that COOJA/MSPSim enables accurate simulation of the power consumption of sensor networks.

Acknowledgements

This work was financed by VINNOVA, the Swedish Agency for Innovation Systems, and the Uppsala VINN Excellence Center for Wireless Sensor Networks WISENET, also partly funded by VINNOVA. This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

References

1. Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: SenSys 2006: Proceedings of the 4th international conference on Embedded networked sensor systems, Boulder, Colorado, USA, pp. 307–320 (2006)
2. Burri, N., von Rickenbach, P., Wattenhofer, R.: Dozer: ultra-low power data gathering in sensor networks. In: IPSN 2007 (2007)
3. Cavin, D., Sasson, Y., Schiper, A.: On the accuracy of MANET simulators. In: Proceedings of the second ACM international workshop on Principles of mobile computing, Toulouse, France (2002)
4. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. IEEE Transactions On Mobile Computing, 272–285 (2004)
5. Colesanti, U.M., Crociani, C., Vitaletti, A.: On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed. In: Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks, Chania, Greece, October 2007, pp. 25–31 (2007)
6. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Workshop on Embedded Networked Sensors, Tampa, Florida, USA (November 2004)
7. Dunkels, A., Österlind, F., Tsiftes, N., He, Z.: Software-based on-line energy estimation for sensor nodes. In: Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV), Cork, Ireland (June 2007)
8. Dutta, P., Culler, D., Shenker, S.: Procrastination might lead to a longer and more useful life. In: Proceedings of HotNets-VI, Atlanta, GA, USA (November 2007)
9. Eriksson, J., Dunkels, A., Finne, N., Österlind, F., Voigt, T.: Mspsim – an extensible simulator for msp430-equipped sensor boards. In: Langendoen, K.G., Voigt, T. (eds.) EWSN 2007. LNCS, vol. 4373. Springer, Heidelberg (2007)
10. Haq, F., Kunz, T.: Simulation vs. emulation: Evaluating mobile ad hoc network routing protocols. In: Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN 2005), London, England (May 2005)
11. Ivanov, S., Herms, A., Lukas, G.: Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In: Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN 2007) (2007)
12. Landsiedel, O., Wehrle, K., Götz, S.: Accurate prediction of power consumption in sensor networks. In: Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II), Sydney, Australia (May 2005)
13. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the first international conference on Embedded networked sensor systems, pp. 126–137 (2003)

14. Lu, G., Krishnamachari, B., Raghavendra, C.: An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. In: International Parallel and Distributed Processing Symposium (IPDPS) (2004)
15. Musaloiu-E., R., Liang, C.-J.M., Terzis, A.: Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In: IPSN 2008 (2008)
16. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with cooja. In: Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), Tampa, Florida, USA (November 2006)
17. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling ultra-low power wireless research. In: Proc. IPSN/SPOTS 2005, Los Angeles, CA, USA (April 2005)
18. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M.: Energy aware wireless microsensor networks. IEEE Signal Processing Magazine 19(2), 40–50 (2002)
19. Schiller, J., Ritter, H., Liers, A., Voigt, T.: Scatterweb - low power nodes and energy aware routing. In: Proceedings of Hawaii International Conference on System Sciences, Hawaii, USA (2005)
20. Shah, R.C., Rabaey, J.M.: Energy aware routing for low energy ad hoc sensor networks. In: Proc. IEEE Wireless Communications and Networking Conference (WCNC) (March 2002)
21. Shnayder, V., Hempstead, M., Chen, B., Allen, G.W., Welsh, M.: Simulating the power consumption of large-scale sensor network applications. In: 2nd International Conference on Embedded Networked Sensor Systems (ACM SenSys) (November 2004)
22. Titzer, B.L., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN) (April 2005)
23. Trathnigg, T., Moser, J., Weisse, R.: A low-cost energy measurement setup and improving the accuracy of energy simulators for wireless sensor networks. In: Proceedings of REALWSN 2008 (April 2008)
24. van Dam, T., Langendoen, K.: An adaptive energy-efficient MAC protocol for wireless sensor networks. In: Proceedings of the first international conference on Embedded networked sensor systems, Los Angeles, California, USA (November 2003)
25. Voigt, T., Österlind, F.: CoReDac: Collision-free command-response data collection. In: 13th IEEE Conference on Emerging Technologies and Factory Automation, Hamburg, Germany (September 2008)

DHV: A Code Consistency Maintenance Protocol for Multi-hop Wireless Sensor Networks

Thanh Dang, Nirupama Bulusu, Wu-chi Feng, and Seungweon Park

Department of Computer Science,
Portland State University,
PO Box 751, Portland, OR, USA
{dangtx,nbulusu,wuchi,spark}@cs.pdx.edu

Abstract. Ensuring that every sensor node has the same code version is challenging in dynamic, unreliable multi-hop sensor networks. When nodes have different code versions, the network may not behave as intended, wasting time and energy. We propose and evaluate DHV, an efficient code consistency maintenance protocol to ensure that every node in a network will eventually have the same code. DHV is based on the simple observation that if two code versions are different, their corresponding version numbers often differ in only a few least significant bits of their binary representation. DHV allows nodes to carefully select and transmit only necessary bit level information to detect a newer code version in the network. DHV can detect and identify version differences in $O(1)$ messages and latency compared to the logarithmic scale of current protocols. Simulations and experiments on a real MicaZ testbed show that DHV reduces the number of messages by 50%, converges in half the time, and reduces the number of bits transmitted by 40-60% compared to DIP, the state-of-the-art protocol.

Keywords: Code consistency, network reprogramming, sensor networks.

1 Introduction

Experience with wireless sensor network deployments across application domains has shown that sensor node tasks typically change over time, for instance, to vary sensed parameters, node duty cycles, or support debugging. Such reprograming is accomplished through wireless communication using reprogrammable devices. The goal of network reprogramming is to not only reprogram individual sensors but to also ensure that all network sensors agree on the task to be performed.

Network reprogramming is typically implemented on top of data dissemination protocols. For reprogramming, the data can be configuration parameters, code capsules, or binary images. We will refer to this data as a *code item*. A node must detect if there is a different code item in the network, identify if it is newer, and update its code with minimal reprogramming cost, in terms of convergence speed and energy.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 327–342, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Network reprogramming must address two main challenges — *node unreliability* and *network dynamics*. In a static network, sensor nodes run on batteries, leading to inconsistent behaviors, with periods of no operation followed by active periods. Radio communication is also known to be unreliable and dynamic with intermittent connectivity and link asymmetry. Sensor networks are becoming more dynamic due to mobility. Mobile sensor nodes might dynamically leave and join a network, requiring that they also vary their tasks dynamically. For instance, a sensor equipped car might be tasked to report carbon emissions within city A but to report temperature in city B. In this case, even if code items are disseminated correctly to all connected nodes during the reprogramming period, some nodes may not have the current code version.

Naturally, this creates the need for a protocol to ensure that all network nodes have the same up-to-date code items, which we refer to as a *Code Consistency Maintenance Protocol* (CCMP). Depending upon the application, the bandwidth and energy consumption of a CCMP can be comparable to sensing, particularly for networks with mobility or large churn. The key requirements for a CCMP are that they:

- ensure all network nodes *eventually have the same updated code*
- enable a node with an old code item to discover a newer code item and update it with *low latency*
- conserve energy and bandwidth
- scale with both the network size and the total number of code items

We represent code items as a set of *(key, version)* tuples. Each *key* uniquely identifies a code item. The corresponding *version* indicates if the code item is old or new (the higher the version, the newer the code). A CCMP allows any node to discover if there is a *(key, version)* tuple in the network with the same key but a different version compared to its own tuple. A natural approach is to allow a node to keep querying its neighbors, to find if there is a new code item by comparing its own tuple to its neighbors tuples. If there are only two nodes and one code item, a node can simply broadcast it own *(key, version)* tuple periodically. In a realistic scenario involving hundreds of dynamic nodes and hundreds of code items, a node should ask *smart questions* at the *right time* to discover if it needs an update. This requires careful CCMP design to reduce code update latency and to conserve both energy and bandwidth.

Previous CCMP protocols like DRIP [1] and DIP [2] incur high latency, high cost and may be complicated. Both DRIP and DIP are built on top of Trickle [3], a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In DRIP, a node randomly broadcasts each of its own *(key, version)* tuples to its neighbors separately. DRIP scales linearly with the total number of items ($O(T)$). DIP improves the total number of messages and latency to $O(log(T))$ by searching for a different *(key, version)* using hashes of the *(key, version)* tuples. A common feature in both approaches is that they try to detect and identify differences of versions as a whole.

In this paper, we propose a new code consistency maintenance protocol called *DHV*. We observe that *if two versions are different, they often only differ in*

a few least significant bits of their version number rather than in all their bits. Hence, it is not necessary to transmit and compare the whole version identifier in the network. DHV aims to detect and identify differences of version-levels for code items with the goal of transmitting much less data in the network compared to DRIP and DIP. The name DHV comes from the three steps in the protocol — Difference detection, Horizontal search, and Vertical search. Each step requires only $O(1)$ total messages and latency with respect to the total number of items. So DHV can identify a difference with as few as 3 transmissions.

The contributions of this paper are as follows:

- We design and implement DHV, a code consistency maintenance protocol for wireless sensor networks (Sections 3, 4 and 5). DHV is simple, intuitive, and easy to adapt to current systems. DHV does a two dimensional search to identify exactly where the different *(key, version)* tuples are. DHV can detect and identify differences in $O(1)$.
- We evaluate DHV using simulations and real-world experiments on a MicaZ testbed, comparing it to state-of-the-art CCMP protocols (Section 6). Our results show that DHV reduces the number of messages by 50%, converges in half the time, and reduces the number of bits transmitted by 40-60%. Potentially, DHV can conserve significant energy compared to DIP, a state-of-the-art CCMP protocol.

2 Related Work

Code consistency management protocols can be thought of as a complement to data (code) dissemination protocols because a CCMP helps to efficiently discover when a node needs updates. Several dissemination protocols like Deluge [4], Sprinkler [5], and MNP [6] were developed to reprogram the whole network by disseminating new binaries. Maté [7] and Tenet [8] break a program into small code capsules or virtual programs and disseminate them to reprogram the sensors. Finally, DRIP [1], DIP [2], and Marionette [9] disseminate configuration parameters that can change sensor tasks.

Early attempts tried to adapt epidemic algorithms [10] to disseminate code updates during specific reprogramming periods. But there is no way for new nodes to discover past updates. If a node is not updated during the reprogramming period, it will never get updated. To discover if a node needs an update, a natural approach is to query or advertise its information periodically. The network as a whole may transmit an excessive and unnecessary number of query and advertisement messages. To address this problem, Levis *et al* [3] developed the Trickle protocol to allow nodes to suppress unnecessary transmissions. In Trickle, a node periodically broadcasts its versions but politely keeps quiet and increases the period if it hears several messages containing the same information as it has. When a difference is detected, the node resets the period to the lowest preset interval. Trickle scales well with the number of nodes and has successfully reduced the number of messages in the network.

Table 1. Complexity Comparison of different CCMP protocols

Protocol	Total cost	Latency
Scan Serial	$O(T)$	$O(T)$
Scan Parallel	$O(T)$	$O(1)$
DIP	$O(Nlog(T))$	$O(log(T))$
DHV	$O(N)$	$O(1)$

However, the total messaging cost of a naive approach using Trickle scales linearly with the total number of code items, according to [2]. Lin *et al* [2] proposed DIP, a dissemination protocol that scales logarithmically with the total number of items. In DIP, a node periodically broadcasts a summary message, containing hashes of its keys and versions. The use of hashing helps detect if there is a difference in $O(1)$. But, once a difference is detected, DIP requires multiple iterations to hone in on the exact code items that have different version numbers. The search is analogous to a binary search in a sorted array. Therefore, DIP has $O(log(T))$ complexity in both the time and the number of messages required to identify an item that needs an update. DIP uses a bloom filter to further improve the search but it requires extra bytes to be included in every summary message. If there are N new items, then the total number of messages is $O(Nlog(T))$.

Ideally, when there are N new items ($N < T$), we would like to transmit just enough information to identify these N items to update. Both Trickle and DIP transmit redundant information, $O(T)$ and $O(Nlog(T))$ respectively, to identify difference in version numbers. However, if two versions differ by even one bit in their binary representation, the two versions are different from each other. Based on this fact, we develop the DHV protocol, which can detect the differences in $O(1)$ complexity in both time and total number of transmitted messages. The total number of messages required to identify which items have newer versions is $O(N)$. [1] Table 1 shows a complexity comparison of different CCMP protocols.

3 Design Philosophy and Assumptions

3.1 Design Philosophy

Bit-level identification: Previous CCMPs have transmitted the complete version number for a code item. We observe that it may not always be necessary to do so. We treat the version number as a bit array, with the versions of all the code items representing a two dimensional bit array. DHV uses bit slicing to quickly zero in on the out of date code segment, resulting in fewer bits transmitted in the network.

Statelessness: Keeping state in the network, particularly with mobility, is not scalable. DHV messages do not contain any state and usually small in size.

[1] Strictly speaking, the DHV cost has one component with $O(T)$. However, the constant is very small and $O(T)$ is often less than 2 messages for T less than 256. Please refer to 4.1 for further details.

Preference of a large message over multiple small messages: To reduce energy consumption, it is better to transmit as much information possible in a single maximum length message rather than transmit multiple small messages. Sensor nodes turn off the radio when they are idle to conserve energy. Radio start-up and turn-off times (300 microseconds) are much longer than the time used to transmit one byte (30 microseconds) [11]. A long packet may affect the collision rate and packet loss. However, that effect only becomes noticeable under bursty data traffic conditions.

3.2 Assumptions

We assume that the order of code items in the set of *(key, version)* tuples is the same at all nodes, by using the same sorting algorithm at all nodes as the keys are often assigned at a base station and are unique. This assumption allows DHV to identify which items need updates from the indices of the different versions. In rare cases, nodes might have a different number of items, requiring that they compare every *(key, version)* tuple to identify which items are missing. We do not address these rare cases here.[2]

We also assume that the version number is incremented by only one for each update. This assumption is reasonable and implicitly made in both DRIP and DIP.[3] It allows us to infer that, if two version numbers are different, they mostly differ in a few least significant bits. DHV exploits this assumption to restrict the comparison scope to only a few least significant bits of the versions.

4 The DHV Protocol

DHV has two main phases — *detection* and *identification*. In detection, each node broadcasts a hash of all its versions called a *SUMMARY message*. Upon receiving a hash from a neighbor, a node compares it to its own hash. If they differ, there is at least one code item with a different version number.

In identification, the *horizontal search* and *vertical search* steps identify which versions differ. In horizontal search, a node broadcasts a checksum of all versions, called a *HSUM message*. Upon receiving a checksum from a neighbor, the node compares it to its own checksum to identify which bit indices differ and proceeds to the next step. In vertical search, the node broadcasts a bit slice, starting at the least significant bit of all versions, called a *VBIT message*. If the bit indices are similar, but the hashes differ, the node broadcasts a bit slice of index 0 and increases the bit index to find the different locations until the hashes are the same. Upon receiving a VBIT message, a node compares it to its own VBIT to identify the locations corresponding to the differing *(key, version)* tuples. After identifying which *(key, version)* tuples differ, the node broadcasts these

[2] We plan to address these cases in the future as we integrate DHV with current dissemination protocols.

[3] In the implementations of DRIP and DIP, each time a code item is updated, its version number is incremented by 1.

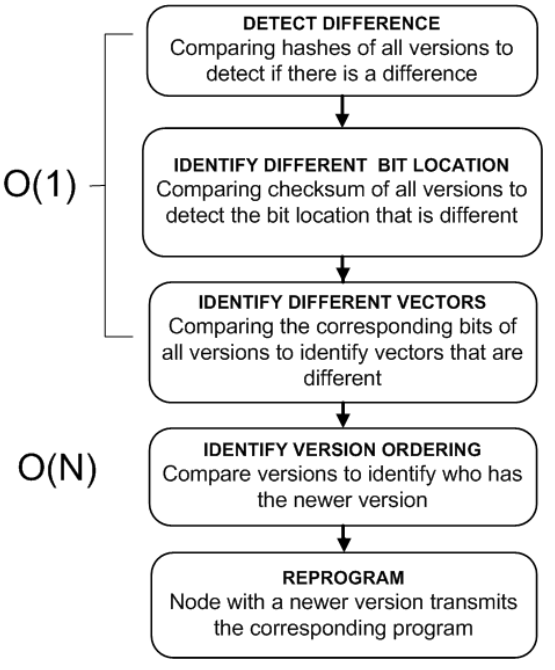


Fig. 1. Main steps in the DHV protocol. DHV completes a search in only three steps, having 0(1) complexity in both latency and cost.

(key, version) tuples in a *VECTOR message*. Upon receiving a VECTOR message, a node compares it to its own *(key, version)* tuple to decide who has the newer version and if it should broadcast its DATA. A node with a newer version broadcasts its DATA to nodes with an older version.

Figure 1 illustrates the DHV protocol steps to complete a search to identify which vectors differ. DHV requires $O(1)$ in both latency and cost to identify all the differences, in contrast to DIP, which requires $O(log(T))$ in both latency and cost for searching.

Figure 2 illustrates how DHV works. Node 1 and Node 2 have a set of keys and versions, in which key number 2 has different versions. Node 1 first broadcasts its SUMMARY hash of all the versions. Node 2 receives hash 1 and sees that hash 1 is different from hash 2 of node 2. Hence, node 2 broadcasts its HSUM message, checksum of all versions. Node 1 receives the HSUM message 2 and compares it to its own checksum. Node 1 identifies that the 2nd bits differ. Hence, node 1 copies the 2nd bit of all the versions into one or more VBIT messages and broadcasts them. Node 2 receives a VBIT message, compares it to its own VBIT message and detects that the 2nd bits differ. Hence, node 2 knows that the item with key index 2 is different from its neighbors. Node 2 broadcasts a VECTOR message containing *(key 2, version 2)*. Node 1 receives *(key 2, version 2)* from node 2 and sees that node 1 has a newer version of this item. Hence, node 1 broadcasts the DATA of key 2.

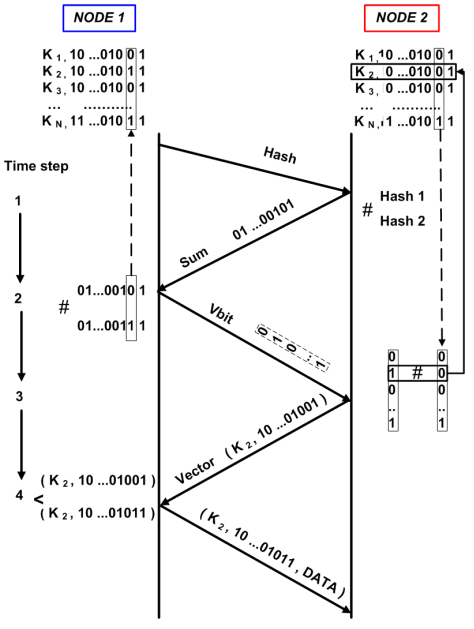


Fig. 2. DHV protocol example. Nodes identify a difference in only 3 steps.

4.1 Message Formats

DHV uses 5 message types (Figure 3). One byte is used to indicate message type.

SUMMARY: This message contains the hash of all versions as well as the random seed used for hashing. It contains the least amount of information. A node can only detect if there is a difference or not using this message.

HSUM: This message contains the checksum of all versions, the hash of all versions as well as the random seed used for hashing. It is used to identify which bit indices differ.

VBIT: This message contains the corresponding bits of all versions and is used to identify which vectors differ. The bits corresponding to the differing indices of the VBIT messages are identified from the HSUM messages. If two HSUM messages are similar but the SUMMARY messages differ, the VBIT messages for the least significant bit indices are compared. This approach is suitable as the versions mostly differ in a few least significant bits. The VBIT size can be varied. DHV prefers large messages over multiple small messages. The VBIT message often has either the maximum packet length or the smallest size that can fit all the corresponding bits of all versions. The size of this message scales linearly with the total number of code items. However, the constant factor is very small. Hence, DHV transmits only a few VBIT messages for searching.

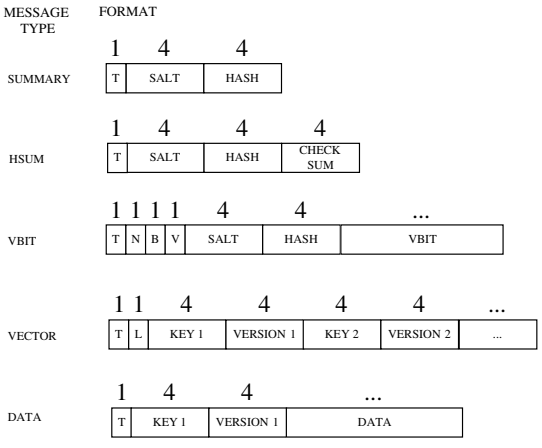


Fig. 3. DHV message format

VECTOR: This message contains one or more *(key, version)* tuples, and is used to identify who has newer versions to and send the code items.

DATA: This message contains the actual code items to be updated.

4.2 Suppression Mechanism

Suppression mechanisms must be carefully designed to avoid flooding the network. As in Trickle, we develop our own suppression mechanism appropriate for DHV. A decision if to send out a message is made when a Trickle timer fires after a specified time interval. When the timer fires, if a node has DATA, VBIT, or HSUM messages to send, it will send one message out, selected by order of importance, DATA, VBIT, and HSUM messages. If there are no DATA, VBIT, HSUM messages to send and the node has heard two or more messages in the last interval, it decides to suppress its own messages. Otherwise, the node will send either the VECTOR or the SUMMARY message. If a SUMMARY message is sent, the next time interval is doubled as it expects that the network is stable.

5 Implementation

We have implemented DHV in TinyOS 2.1 and tested the protocol with MicaZ motes. DHV uses the Trickle timer [7] to control transmission suppression. Similar to DIP, both key and version in DHV are 4 bytes. The key and version sizes can be adjusted for the application. Table 2 compares DHV memory usage to DIP. They are largely similar, but DHV uses more RAM than DIP. We plan to carefully optimize the code implementation and believe that the RAM usage can be reduced significantly in subsequent versions.

Table 2. DHV Implementation Statistics

Item	DIP	DHV
ROM (Byte)	20K	19K
RAM (Byte)	534	8739
Compiled code size (Byte)	63K	81K

6 Experimental Design and Analysis

6.1 Goals and Metrics

Previous work has shown that DIP outperforms other CCMP protocols [2]. Our experimental goals are to study if DHV performs better than the state-of-the-art DIP protocol over different scenarios. We use the following metrics to evaluate the performance, with a lower value indicating better performance for all metrics.

- Total latency to update new items. This metric indicates how fast a CCMP can help the network converge.
- Total numbers of transmitted messages and transmitted bytes to update new items. These metrics indirectly represent energy and bandwidth consumption.
- Total number of transmitted DATA messages. DATA can be as small as a few bytes or as big as several KBytes.
- Latency of the first transmitted DATA message. This metric indicates how quickly a new item is discovered.

6.2 Methodology

We conducted both simulations using TOSSIM [12], a discrete event simulator tool for wireless sensor networks, and experiments using our DHV implementation on a real MicaZ sensor network testbed (Figure 4 Left).

TOSSIM does not allow simulation of packet loss directly. Instead, the packet loss depends on several parameters like receiving gain, noise, and clear channel access threshold. As a first step, we studied the effect of these parameters on packet loss. We simulated a two-node network. The noise is simulated using the state-of-the-art closest pattern matching approach with noise traces from the Stanford Meyer library, available in the TinyOS source code. Due to memory limitations, we only use the first 1000 entries in the trace, which is well above the minimum requirement of 100 entries. Figure 4 (Right) shows the packet loss rate versus receiving gain. The receiving gains correspond to packet loss rates of 5, 10, 15, 20, 25, 30, 35, 40, and 45% are -70, -74, -76, -78, -81, 84, -87,-88, -89dBM, respectively. We conducted three sets of experiments. Two sets are simulation-based and one is on a real MicaZ sensor network. The first set studies how DHV performance is impacted by different parameters including the total number of items T, the total number of new items N, the packet loss rate L, and the density D. Density refers to the number of radio communication neighbors. We compare DHV and DIP in a clique network. The default setting is $D =$

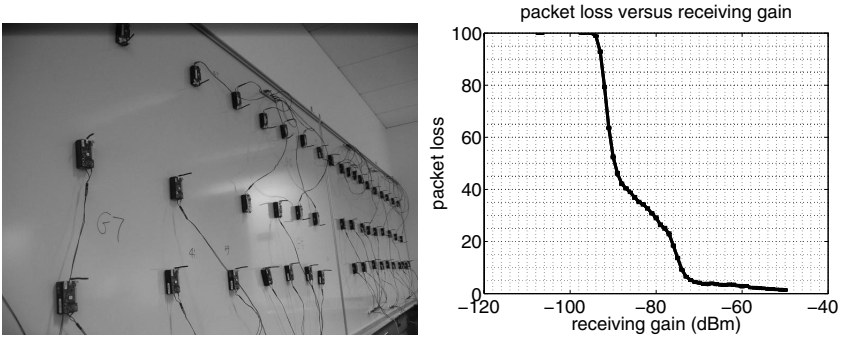


Fig. 4. (Left) Real MicaZ Testbed. (Right) Packet loss versus receiving gain using TOSSIM simulation.

32 (nodes), $T = 64$ (keys), $N = 8$ (keys), $L = 5\%$. We vary D, T, N, and L and observe the total numbers of transmitted messages, transmitted bytes, and DATA messages, the true dissemination time, and the latency until the first data is sent out. Each experiment is repeated 10 times to account for randomness in timing. Like DIP, DHV uses 2 key/version tuples per VECTOR message to ensure comparability.

The second set includes two experiments with a medium and a tight density multi-hop network. The total number of nodes is 225. The topology and link configurations are extracted from example files in TOSSIM (15-15-medium-mica2-grid.txt and 15-15-tight-mica2-grid.txt in tossim/topologies directory). We observe the total number of transmitted messages and bytes required to complete updating as well as the update progress in terms of time.

Finally, we experimented with a real MicaZ sensor network testbed. We varied the number of sensor nodes and observed the total number of transmitted messages and total time required to complete updating the whole network. Unfortunately, the latest version of DIP on TinyOS 2.1 did not function properly on the MicaZ platform. Due to time constraints and lack of hardware, the TinyOS maintainers were unable to verify the problem. Hence, we can only evaluate DHV experimentally, and not compare it to DIP.

6.3 Experimental Results

DHV Performance versus Total Number of Items. Figure 5 compares DHV and DIP when we vary the total number of items, T. DHV performance is relatively constant with T. Meanwhile, the cost and latency in DIP increases as T increases. As an example case, when $T = 64$, nodes using DHV transmit only about 40% of the total number of messages and complete reprogramming within 45% of the time taken by nodes using DIP. The discovery latency for the first new item in DHV is also a constant. Nodes using DHV also transmit only about 50% of the total number of DATA messages and less than 50% of the total number of bytes compared to nodes using DIP.

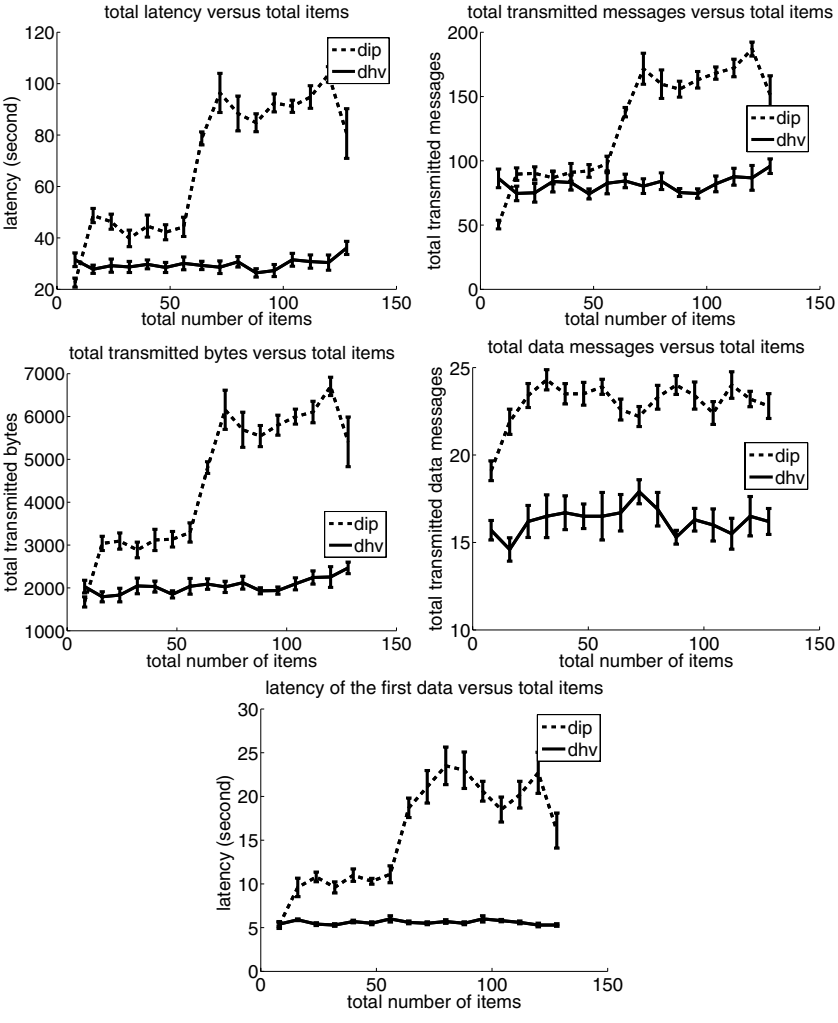


Fig. 5. Total latency versus total items: $D = 32$, $N = 8$, $L = 5\%$. T varies from 8 to 128.

DHV Performance versus Total Number of New Items. Figure 6 compares DHV and DIP when we vary the number of new items. DHV always uses fewer messages than DIP and uses only half the time to complete updating the network. For example, when $N = 32$, DHV uses about 70% of the messages of DIP and completes reprogramming in half the time.

DHV Performance versus Network Density. Figure 7 compares DHV and DIP when we vary the density of the network. DHV completes updating twice faster and uses 50% fewer messages than DIP. Since, DHV uses smaller size

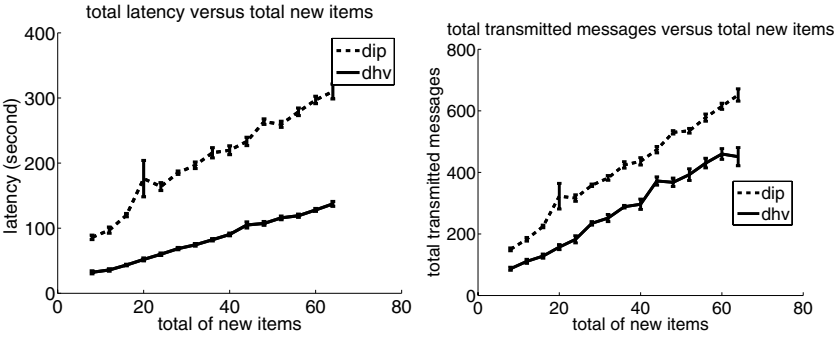


Fig. 6. Total latency versus total new items: $D = 32$, $T = 64$, $L = 5\%$. N varies from 8 to 64.

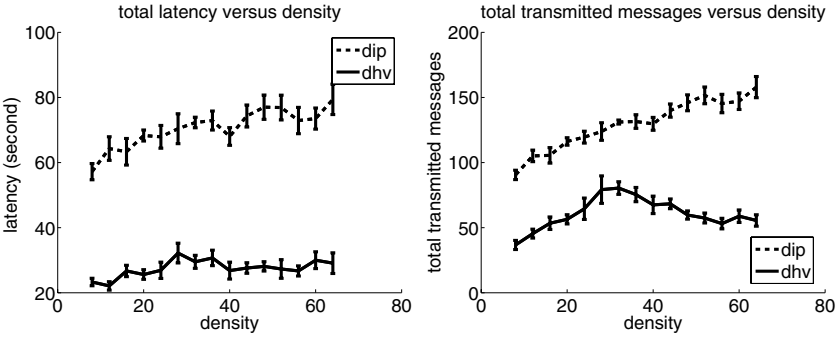


Fig. 7. Total latency versus network density: $T = 64$, $N = 8$, $L = 5\%$. D varies from 8 to 64.

messages than DIP except for the VBIT message, the total transmitted bytes in DHV are also much smaller than DIP.

DHV Performance versus Packet Loss Rate. Figure 8 compares DHV and DIP when we vary the packet loss rate. DHV completely outperforms DIP in terms of latency. DHV completes updating ten times faster than DIP while using only half the number of messages. Unlike previous experiments, DHV and DIP use similar numbers of DATA messages. But both DHV and DIP exhibit a high performance variation, that increases with the packet loss rate.

DHV Performance in Multi-hop Networks. Figure 9 shows the total number of transmitted messages and bytes to complete updating a multi-hop network using DHV and DIP. As expected, a denser network requires fewer total number of messages for updating data. In both medium and tight density networks, DHV uses about 60% to 70% number of messages and about 50% number of bytes compared to DIP. Figure 10 plots the update progress time versus the number

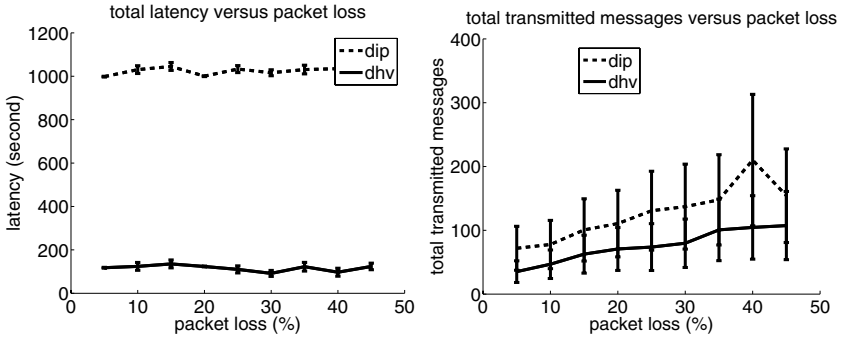


Fig. 8. Total latency versus network density: $D = 32$, $T = 64$, $N = 8$. Packet loss rate L varies from 5% to 45%.

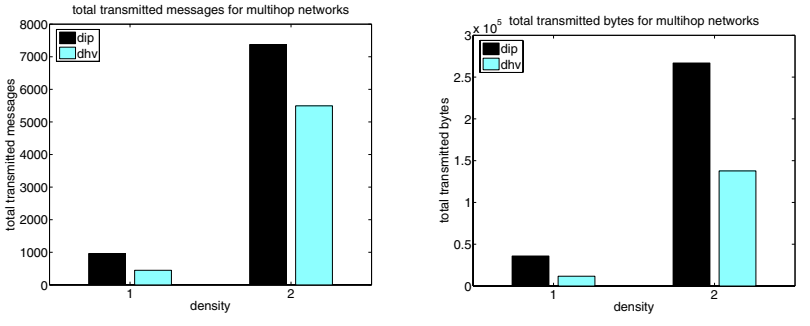


Fig. 9. Total transmitted messages for multi-hop networks

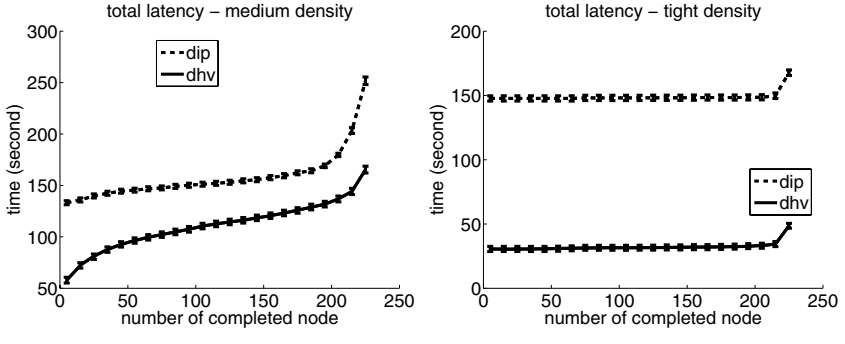


Fig. 10. Convergence time for multi-hop networks

of completed nodes for both DHV and DIP. In update completion time, DHV is at least two times faster than DIP in medium density networks and nearly five times faster in tight density networks. In the medium density network, both

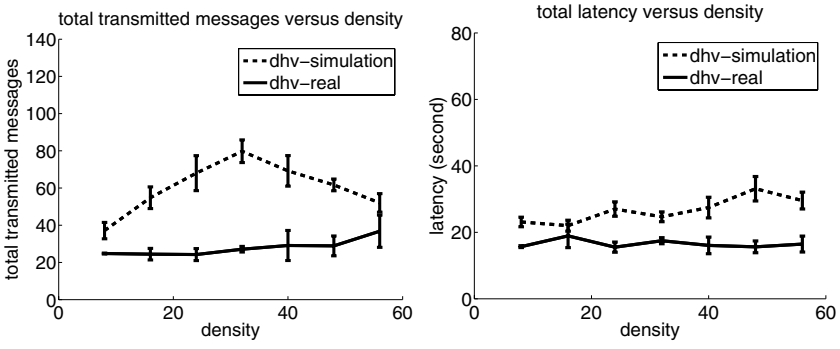


Fig. 11. Total transmitted messages versus network density: $T = 64$, $N = 8$, D is varied from 8 to 56 nodes

DHV and DIP update completion times grow linearly with the completed nodes because in each transmission, only a few nodes can receive the messages. The update progresses from the node with the new items and spreads out to the whole network. Hence, the number of completed nodes grows linearly with time. In contrast, in the tight density network, when a node broadcasts, most other nodes receive the message. Hence, the network converges quickly. The inflection point when the number of completed nodes is around 200 can be explained by the fact that some nodes always receive messages with high noise. It takes much longer to complete updating these nodes.

DHV Performance on a real MicaZ test-bed. Figure 11 shows the total number of transmitted messages and the update completion time on a real MicaZ network. There are total T=64 items and N=8 new items. The number of nodes is varied from 8 to 56 nodes. Surprisingly, DHV performs better in the real testbed than the simulation in section 6.3. DHV uses both fewer total messages and completes updating earlier in the real testbed than in the simulation. This might be explained by the fact that in the simulation the packet loss rate is 5% while the packet loss rate of the testbed is pretty low.

7 Discussion

Unlike dissemination protocols like Deluge or MNP [4,6], which address the problem of how to disseminate code items in the network, code consistency management protocols address the problem of when to perform code updates. Therefore, it is important to ensure that the two protocols can be integrated well. The DHV protocol enables a dissemination protocol to decide when to begin transferring a new code item in the network so that the network can be updated quickly while still conserving energy. This decision making process can be independent from the operation of the dissemination protocol. Hence, the DHV protocol should work well with the existing dissemination protocols. We plan to integrate the DHV protocol with Deluge in the next source code release. Sensor networks are becoming

more heterogeneous with devices based on different hardware/software platforms, having different numbers of code items. Ensuring code consistency for such networks is challenging, and a future research topic.

8 Conclusion

We have proposed and evaluated the DHV protocol to maintain code consistency in wireless sensor networks. The key innovation in DHV is that it reduces the number of transmitted bytes in the network by carefully selecting and transmitting only absolutely necessary information at the bit level to detect and identify which code items need updates. Together with a carefully designed suppression mechanism, DHV is able to reduce the total number of messages significantly. Theoretically, DHV can identify differences with $O(1)$ complexity in the total number of items instead of logarithmically compared to DIP. Simulations and real-world experiments validate that DHV performs at least twice better than the state-of-the-art DIP protocol. We believe that DHV can not only be used in wireless sensor networks but also in other distributed applications that require data consistency. The preliminary version of the DHV source code can be downloaded at `http://sys.cs.pdx.edu/home/dhv`.

Acknowledgments. This research was supported in part by National Science Foundation grants 0514818 and 0722063. Any opinions, findings, conclusions or recommendations it contains are those of the authors, and do not necessarily reflect the views of the Foundation.

References

1. Tolle, G., Culler, D.: Design of an application-cooperative management system for wireless sensor networks. In: Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005), Istanbul, Turkey (2005)
2. Lin, K., Levis, P.: Data discovery and dissemination with dip. In: Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (IPSN 2008), Washington, DC, USA, pp. 433–444. IEEE Computer Society Press, Los Alamitos (2008)
3. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI 2004), Berkeley, CA, USA, p. 2. USENIX Association (2004)
4. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd international conference on Embedded networked sensor systems (Sensys 2004), pp. 81–94. ACM, New York (2004)
5. Naik, V., Arora, A., Sinha, P., Zhang, H.: Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices. In: Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS 2005), Washington, DC, USA, pp. 277–286. IEEE Computer Society Press, Los Alamitos (2005)

6. Kulkarni, S.S., Wang, L.: Mnp: Multihop network reprogramming service for sensor networks. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005), Washington, DC, USA, pp. 7–16. IEEE Computer Society Press, Los Alamitos (2005)
7. Levis, P., Gay, D., Culler, D.: Active sensor networks. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI 2005), Berkeley, CA, USA, pp. 343–356. USENIX Association (2005)
8. Gnawali, O., Jang, K.Y., Paek, J., Vieira, M., Govindan, R., Greenstein, B., Joki, A., Estrin, D., Kohler, E.: The tenet architecture for tiered sensor networks. In: Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys 2006), pp. 153–166. ACM Press, New York (2006)
9. 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: Proceedings of the fifth international conference on Information processing in sensor networks (IPSN 2006), pp. 416–423. ACM, New York (2006)
10. Akdere, M., Bilgin, C.Ç., Gerdaneri, O., Korpeoglu, I., Ulusoy, Ö., Çetintemel, U.: A comparison of epidemic algorithms in wireless sensor networks. Computer Communications 29(13-14), 2450–2457 (2006)
11. Kramer, M., Geraldy, A.: Energy measurements for micaz node. Technical Report, Technical University Kaisers Lautern,GI/ITG KuVS, 1–7 (2006)
12. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st international conference on Embedded networked sensor systems (Sensys 2003), pp. 126–137. ACM Press, New York (2003)

Sensornet Checkpointing: Enabling Repeatability in Testbeds and Realism in Simulations

Fredrik Österlind, Adam Dunkels, Thiemo Voigt,
Nicolas Tsiftes, Joakim Eriksson, and Niclas Finne

Swedish Institute of Computer Science
{fros,adam,thiemo,nvt,joakime,nfi}@sics.se

Abstract. When developing sensor network applications, the shift from simulation to testbed causes application failures, resulting in additional time-consuming iterations between simulation and testbed. We propose transferring sensor network checkpoints between simulation and testbed to reduce the gap between simulation and testbed. Sensornet checkpointing combines the best of both simulation and testbeds: the non-intrusiveness and repeatability of simulation, and the realism of testbeds.

1 Introduction

Simulation has proven invaluable during development and testing of wireless sensor network applications. Simulation provides in-depth execution details, a rapid prototyping environment, nonintrusive debugging, and repeatability. Before deploying an application, however, testing in simulation is not enough. The reason for this, as argued by numerous researchers, is over-simplified simulation models, such as the simulated radio environment. To increase realism, testbeds are employed as an inter-mediate step between simulation and deployment. Testbeds allow applications to be run on real sensor node hardware, and in realistic surroundings. Migrating an application from simulation to testbed is, however, expensive [8,21,26]. The application often behaves differently when in testbed than in simulation. This causes additional time-consuming iterations between simulation and testbed.

To decrease the gap between simulation and testbed, hybrid simulation has been proposed [12, 15, 23, 24]. Hybrid simulation contains both simulated and testbed sensor nodes. Although hybrid simulation is feasible for scaling up networks – to see how real nodes behave and interact in large networks – it does not benefit from many of the advantages of traditional simulation. Testbed nodes do not offer nonintrusive execution details. Tests are not repeatable due to the uncontrollable testbed environment: the realism, one of the main advantages of hybrid simulation. Finally, test execution speed is fixed to real-time; it is not possible to increase the test execution speed as when with only simulated nodes.

We propose a drastically different approach for simplifying migration from simulation to testbed. In our approach every node exists both in testbed and simulation. We checkpoint and transfer network state between the two domains.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 343–357, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The entire network is at any given time, however, executing only in either simulation or testbed.

Our approach benefits from advantages of both simulation and testbeds: non-intrusive execution details, repeatability, and realism. By transferring network state to simulation, we can benefit from the inherent properties of simulation to non-intrusively extract network details. We can deterministically repeat execution in simulation, or repeatedly roll back a single network state to testbed. Finally, we benefit from the realism of real hardware when the network is executed in testbed.

To highlight benefits of moving network state between simulation and testbed, consider the following example. A batch-and-send data collection application is executing in a testbed, and is periodically checkpointed by the testbed software. One of the network checkpoints is imported into simulation, and so the simulator obtains a *realistic* network state. While in simulation, network details can be *non-intrusively* extracted and altered. Extracting details this way is comparable to having a hardware debugging interface, such as JTAG, to every node in the testbed: the full state is available without explicit software support on the nodes. In contrast to with hardware debugging interfaces, however, network execution may be continued in simulation.

Simulation provides good debugging and visualization tools, but may not accurately measure environment-sensitive parameters such as multi-hop throughput. Hence, when a collector node is about to send its batched data, the network state is moved back to testbed for accurately measuring the bulk transfer throughput on real hardware. Note that this checkpoint can be imported into testbed multiple times, *repeating* the same scenario, resulting in several throughput measurements.

The rest of this paper is structured as follows. After introducing our checkpointing approach and its application areas in Section 2, we implement it in Section 3. The approach is evaluated in Section 4. Finally, we review related work in Section 5, and conclude the paper in Section 6.

2 Sensornet Checkpointing

Sensornet checkpointing is mechanism for extracting network state from both real and simulated networks. A network checkpoint consists of the set of all sensor node states. Extracted node states can be stored local to each node for offline processing, such as in external flash. When network execution is finished, node states are extracted from external flash to be analyzed. Node state can also be transferred online to outside the network via radio or serial port. A network rollback, the opposite of checkpointing, restores a previously extracted state to the network. Both checkpointing and rolling back network state can be performed at any time, and is synchronous: states from all network nodes are extracted and restored at the same time. During checkpoint and rollback operations, the network is frozen, so the operations are nonintrusive to regular network execution. Figure 1 demonstrates network checkpointing.

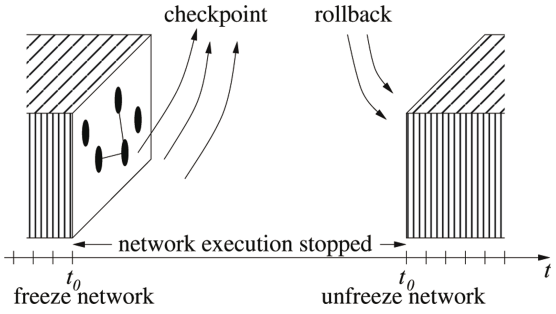


Fig. 1. Checkpointing freezes all nodes in the network at a given point in time. The state of all network nodes can be either stored on the individual nodes for offline processing, or directly downloaded to an external server.

Although sensornet checkpointing is a general mechanism, in this paper we focus only on online checkpointing via wired connections. Freezing and unfreezing networks is implemented by serial port commands that stop respectively start node execution. Individual node states are transferred only when the entire network is frozen. Checkpoints could be transferred over radio links for use in deployed networks without wired connections. In this paper we only consider transferring network state between testbeds and simulation.

Node state can be analyzed to extract detailed execution details, such as internal log messages, past radio events, or energy consumption estimates. A node state can also be altered before rolling back the network state. By altering a node state, the node can be reconfigured or injected with faults.

Checkpointing is performed on both simulated and real networks, and a network checkpoint can be rolled back to either simulation or testbed. Hence, by checkpointing in one domain and rolling back to another, we transfer the network state between the domains, for example from testbed to simulation. Since checkpointing is performed non-intrusively (all nodes are frozen), we benefit from advantages associated with both simulation and testbed. For example, we can use powerful visualization and debugging tools available only in simulation on a network state extracted from testbed.

Network checkpointing can be integrated into testbed software. The software periodically checkpoints the network, and stores the state to a database. The checkpointing period is application specific, and may range from seconds to hours. By rolling back the same network state several times, testbed runs can be repeated. This is useful for rapidly evaluating interesting phases during a network lifetime. For example, this approach can be used to measure how many radio packets are required to re-establish all network routes after a node malfunction, or the time to transfer bulk data through a network.

When migrating an application from simulation to testbed, checkpointing can be used to study and compare results of the different application phases. In a data collection network, the first application phase is to setup sink routes on all collector nodes. Later application phases include collectors sending sink-oriented data,

and repairing broken links. By executing only the setup phase in both simulation and testbed, and compare the two resulting network states, a developer can make sure that this phase works as expected in testbed. We see several sensor network applications that benefit from sensornet checkpointing: visualization, repeating testbed experiments, protocol tuning, fault injection, simulation model validation, and debugging.

Testbed visualization. Checkpoints contain application execution details of all nodes in the network. By rolling back a testbed checkpoint to simulation, information such as node-specific routing tables, memory usage, and radio connections can be visualized using regular simulation tools. The traditional approach for testbed visualization is to instrument sensor network applications to output relevant execution details, for example by printing the current node energy consumption on the serial port. This may have a significant impact on the application, for example by affecting event ordering. The application impact of visualization via sensornet checkpointing is lower since the network is frozen when the execution details are extracted.

Repeated testbed experiments Testbed software can be customized to checkpoint the network when it is about to enter a pre-determined interesting phase. By rolling back the network state, interesting phases can be repeated in testbed, enabling targeted evaluations. Advantages of using checkpointing for evaluations are less test output variance, and faster test iterations. We get less variance in test results since the same network setup is used repeatedly. The test iterations are shorter because the network repeats *only* the evaluated execution phase.

Automated protocol tuning. Repeated testbed experiments can be extended by modifying parameters between each test run. In simulation, parameters can be modified using regular simulation tools. Techniques for automatically tuning parameters, such as reinforcement learning, are often used in simulation only due to the many iterations needed. With checkpointing, we enable use of reinforcement learning in testbeds: initial learning is performed in simulation, and the system is further trained in testbed.

Fault injection in testbed. Fault injection is a powerful technique for robustness evaluations. To inject errors in a testbed, a testbed checkpoint is rolled back to simulation. Errors are injected in simulation, for example processor clock skews, dropped radio packets, or rebooted sensor nodes. The network state is then again checkpointed and moved back to testbed. Fault injection helps us answer questions starting with "what would happen if. . . ".

Simulation model validation. Step-by-step comparisons of testbed and simulation execution help us validate and tune simulation models.

Debugging testbeds. Debugging is a challenging task in wireless sensor networks [17, 27]. Debugging nodes is difficult due to the distributed nature of sensor networks in combination with the limited memory and communication bandwidth of each sensor node. Moreover, debugging approaches that depend

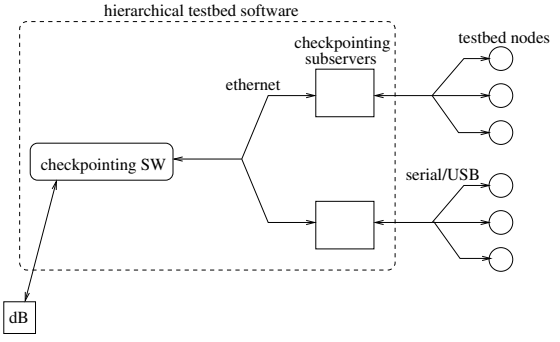


Fig. 2. The checkpointing software is hierarchical and connects the main server to all testbed nodes

on radio communication may be too intrusive, or may not even work in a faulty network. In contrast, simulation is well suited for debugging sensor networks. In simulation, a developer has full control of the entire network, can stop and restart the simulation, and can perform node-specific source-level debugging. Sensornet checkpointing can be used for debugging in simulation, whereas regular network execution is on real hardware. When an error is discovered in testbed, the network is checkpointed and rolled back to simulation. Simulation tools may help expose the error cause. In addition, if the testbed is periodically checkpointed, an earlier checkpoint rolled back to simulation may re-trigger the testbed error in simulation.

3 Implementation

We implement checkpointing support on Contiki [3] and the Tmote Sky sensor nodes [16]. The Tmote Sky is equipped with a MSP430F1611 processor with 10Kb RAM, 1Mb external flash, and an IEEE 802.15.4 Chipcon CC2420 packet radio [2]. The sensor nodes communicate using the serial port. The node checkpointing implementation consists of a single process that handles received commands, and forwards state requests to device drivers. The process is executed in a thread separate from the surrounding Contiki. The thread has a 128 byte stack, which is not included in the node state. We use Contiki's cooperative multi-threading library, however, the checkpointing process does not rely on Contiki-specific functionality. Hence, the node checkpointing implementation should be easily ported to other sensor network operating systems.

Devices support checkpointing by implementing two functions: one for extracting device state and one for restoring device state. We implement state support in four different devices: the LEDs, the radio, the microcontroller hardware timers, and the memory device handling RAM.

See Figure 2 for an overview of the testbed checkpointing software. We implement the subserver on the ASUS WL-500G Premium wireless router [6]. The

Tmote Sky sensor nodes are connected to the routers' USB ports, and each router is connected via ethernet to the local network. The routers run OpenWRT [13], a Linux distribution for embedded devices. For accessing sensor node serial ports, we use the Serial to Network Proxy (ser2net) application [19]. ser2net is an open source application for forwarding data between sockets and serial ports.

For checkpointing simulated networks, we use the Contiki network simulator COOJA [14]. COOJA is equipped with the MSP430 emulator MSPSim [5]. MSPSim supports emulating Tmote Sky sensors. Note that checkpointing is performed via serial ports both in real and simulated networks.

3.1 Network State

We define network state as the set of all node states. Node state consists of:

Firmware. The firmware programmed on each node, i.e. the compiled program code. Unless the program is altered during run-time, the firmware does not need to be included in the node state.

External flash. The 1mb external flash. Unless the external flash is altered during run-time, it does not to be included in the node state.

Memory. The 10kb RAM memory. Memory state captures stack and variables.

Timer system. Hardware timers, both configuration and current timer values. Note that software timers are included in the memory state.

LEDs. The node's light-emitting diodes: on or off.

Radio. The CC2420 radio state includes whether the radio is in listen mode, or turned off. We do not include CC2420 chip buffers in the radio state.

Although our definition of node state can be extended with more devices, the definition is in accordance to the needs we observed during this work. The definition should be easy to extend to include more components.

3.2 Communication Protocol

The communication protocol between the checkpointing software and the sensor nodes consists of three asynchronous phases: *freeze node*, *execute commands*, and finally *unfreeze node*. The three phases are asynchronous for better inter-node synchronization: all nodes are frozen before the execute command phase is started on any of the nodes. Similarly, the *unfreeze node* phase is not initiated on any node until the *execute commands* has finished on all nodes.

The *freeze node* phase is initiated by sending a single byte command to a node. The checkpointing process, executing in the serial port interrupt, immediately performs two tasks: it disables interrupts and stops hardware timers. The node then enters the *execute command* phase. The *execute command* phase

consists of the sensor node handling commands from the checkpointing software. Two commands are available: SET_STATE, and GET_STATE. The SET_STATE command prepares the node to receive node state. The GET_STATE command instructs the node to serialize and transmit its current state. When the node has finished handling commands, the *unfreeze* phase is initiated. This phase again starts hardware timers and enables interrupts.

Since the checkpointing process is executed in the serial port interrupt handler, regular node execution is disabled until the node is unfrozen. A limitation of this implementation is that the serial port interrupt has to wait for other currently executing interrupt handlers to finish. Hence, a node stuck in another interrupt handler cannot be checkpointed.

3.3 Checkpointing Thread

Checkpointing runs in its own thread. The checkpointing thread preempts the operating system to perform a checkpoint. The node state includes the operating system stack, but not the checkpointing thread stack. When restoring node memory, we do not overwrite the checkpointing thread stack.

When the checkpointing process exists after rolling back memory state, the operating system is changed including the program counter (PC) and stack pointer (SP). Figure 3 shows an overview of how the operating system memory is restored from the checkpointing thread. Figure 4 contains pseudo code of the serial port interrupt handler, the checkpointing process, and device driver checkpointing support.

3.4 Node State in Simulation

Simulated nodes extract and restore state using the same checkpointing process as real nodes. Since simulated nodes emulate the same sensor node firmware as real nodes, the same checkpointing software can be used to interact with both real and simulated nodes.

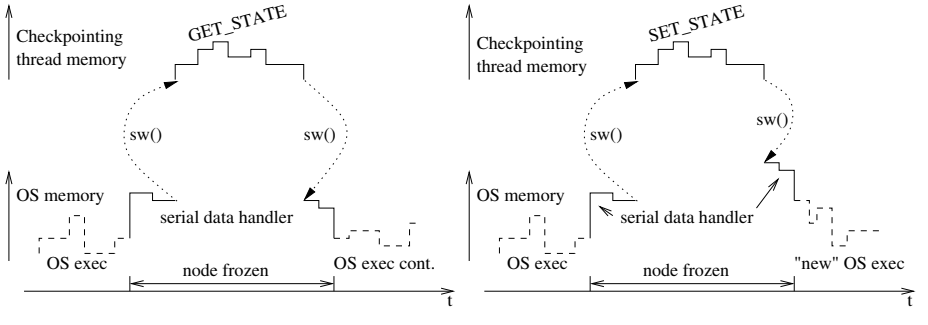


Fig. 3. The checkpointing process is run in a separate thread to enable checkpoints of the operating system stack memory. Left: after checkpointing node memory, control is returned to the OS. Right: after rolling back node memory, the operating system has changed.

```
/* INTERRUPT HANDLER: Serial data */
handle_serial_data(char c)
  freeze_system(); /* Disable timers, interrupts */
  sw(checkpointing_process); /* Switch to checkpointing thread (BLOCKS) */
  unfreeze_system(); /* Enable timers, interrupts */

/* Checkpointing process blocking reads serial data */
checkpointing_process()
  cmd = READ_COMMAND(serial_stream);
  if (cmd CHECKPOINT)
    foreach DEVICE
      DEVICE_get(serial_stream);

  if (cmd ROLLBACK)
    foreach DEVICE
      DEVICE_set(serial_stream);

  if (cmd not UNFREEZE)
    repeat;

/* Devices drivers handle their own state */
mem_set(serial_stream)
  mem[] = read(serial_stream, mem_size);

radio_get(serial_stream)
  write(serial_stream, radio_state[], radio_size);
```

Fig. 4. Checkpointing pseudo code

3.5 Hierarchical Network Freezing

The main server is able to directly communicate with all individual testbed sensor nodes via ser2net. To improve checkpointing synchronization – avoiding parts of the network still executing after other nodes have frozen – we modify the ser2net application to allow sending freeze and unfreeze commands to all nodes connected to a single router. For this purpose, we use ser2net's control port, used for allowing remote control to active ser2net connections. We add support for two new commands: *freeze_all* and *unfreeze_all*. Both commands act on all connected sensor nodes.

3.6 Lost Serial Data

Serial data bytes are sometimes lost when copying state to and from multiple nodes simultaneously. To avoid this problem, we copy state to at maximum one node per subserver. This increases the overall time to transfer network state. It does not, however, affect the inter-node synchronization since all nodes are frozen during the execute command phase. In our current implementation, no CRC check is used to ensure that checkpointed state was transferred correctly. Apart from lost data byte, we have not observed transmission errors in our experiments.

3.7 Hardware Timer Bug

We encountered what appears to be an undocumented MSP430 hardware bug in the timer system during the implementation. In Contiki, the 16-bit hardware

Timer_A is configured for continuous compare mode. A compare register is configured and continuously incremented to generate a periodic clock pulse which drives the software time. According to the MSP430 User's Guide, the interrupt flag is set high when the timer value counts to the compare register.

While stress testing our system with incoming serial data, we observed that occasionally the timer interrupt was set high before the timer value incremented to equal the compare register, i.e. the *timer interrupt handler could start executing before the configured time*. A workaround to the problem is modifying the Timer_A interrupt handler to blocking wait until the timer value reaches the compare register if, and only if, the timer value is equal to the compare register minus 1. Although we have not observed it, we expect similar behavior from Timer_B.

4 Evaluation

We evaluate the sensornet checkpointing intrusiveness by checkpointing a data collection network. Furthermore, we evaluate testbed repeatability by rolling back a single network state and comparing multiple test runs. Finally, we report on the synchronization error of our implementation when checkpointing a network.

In this work, there are several artifacts of our experimental setup that influence results. We checkpoint all nodes sequentially, which can be a time-consuming operation in large networks. Furthermore, we transfer the state uncompressed: checkpointing requires copying more than 10kb per node. Using relatively simple techniques, we could reduce the checkpoint state transfer times. Since the memory is the major part of node state, and since most of the memory is often unchanged between checkpoints, diff-based techniques may significantly reduce node state size. A similar approach is using run-time compression algorithms on the nodes. To evaluate the impact of compression, we run-length encode a number of checkpointed node states. Run-length encoding eliminates long runs of zero-initialized memory. The average size of run-length encoded node states was 7559 bytes, a 26% reduction in size. Note that in this evaluation we only run-length encode node state offline – to use compression or diff-based techniques on real nodes, the node checkpointing process needs additional functionality.

4.1 Intrusiveness: Checkpointing a Data Collection Network

To evaluate the impact of checkpointing a testbed, we implement a simple data collection network. The single-hop network consists of 5 collector nodes and a sink. We perform checkpointing on the collector nodes only. Each collector node samples sensor data, and transmits it via radio to the sink. The sampling intervals, however, differ between the collector nodes. The different send rates for the 5 nodes are: 5, 4, 3, 2, and 1 times per second.

The sink node counts the received radio packets from each collector node. Each radio packet has a sequence number reset at every test run. When the sink

receives a radio packet from collector node 1 with a sequence number equal to or above 150, it decides the test run completed, and logs the test run results. Note that the sink may not have received all 150 radio packets from node 1 to end the test; some radio packets may have been lost in transmission.

We checkpoint the network at different rates during each test run. Checkpointing consists of freezing the network, downloading network state, and finally unfreezing the network. Each test run takes approximately 30 seconds network execution, i.e. without the overhead of checkpointing.

We vary the checkpointing interval on the data collection network. With our test setup, checkpointing less than once every 5 seconds had no impact on the transmitted radio packets. When checkpointing more often, we see an increase in lost radio packets. We believe this is due to the high checkpointing rate combined with not storing radio chip buffers: checkpointing when the sensor node radio chip is communicating may cause a radio packet to be dropped. Figure 5 shows the checkpointing impact on the data collection network. Note that Node 1 transmits a packet every second, and Node 5 transmits a packet every 5 seconds.

We believe the observed checkpointing impact on radio communication can be lessened by including radio chip buffers in the radio state. The current implementation clears the CC2420 radio chip buffers, but includes the radio driver state. If the node is checkpointed when the radio driver has copied only the first half of a radio packet, the radio chip buffers will be empty when rolled back, whereas the radio driver will keep copying the second half of the radio packet. The CC2420 radio chip buffers can be accessed via the radio driver. By including these in the radio state, a radio driver can be checkpointed while communicating with the CC2420 without destroying packet data. However, note also that checkpointing as often as every second is not neccessary in any of the applications discussed in Section 2.

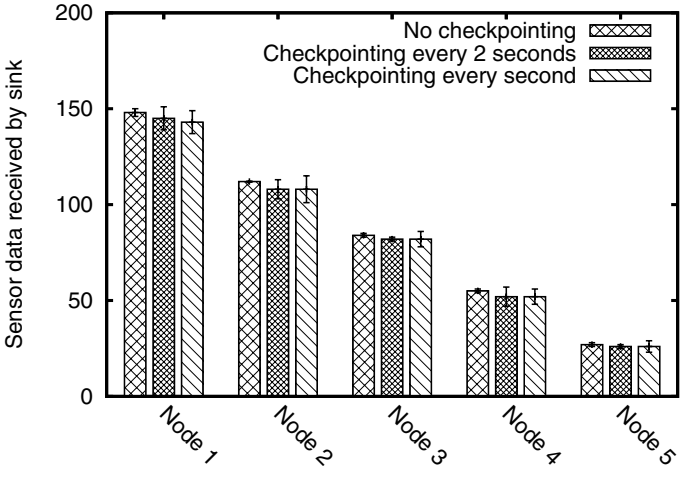


Fig. 5. Checkpointing has little impact on the data collection network, even when checkpointing every second

4.2 Repeatability: Restoring State of a Pseudo-random Network

For evaluating testbed repeatability, we implement a pseudo-random testbed network with 10 sensor nodes. Each node broadcasts radio packets at pseudo-random intervals, on the order of seconds. A radio sniffer outside the testbed records any received packets from the 10 nodes.

During the testbed execution we checkpoint the network once. The network state is then repeatedly rolled back to the network 10 times. The radio sniffer records and timestamps any received packets during a test run. The timestamp resolution is milliseconds. We compare the radio sniffer logs of each test run to evalute testbed repeatability. We perform two experiments using the testbed setup: with and without radio Clear-Channel Assessement (CCA).

By analyzing the sniffer logs, we see that the network execution was repeated when the network state was rolled back, i.e. the ordering of radio packets received by the sniffer node was the same for all test runs. With CCA enabled, we observe that not all packets were transmitted in every test run. This is due to the CCA check failing, an event occasionally occurring in a real testbed. Using the unique sequence number of each received packet, we can calculate the average arrival time of each packet. Figure 6 shows the average number of packets received in a test run, and each packet arrival time deviation from mean as measured by the sniffer node. The arrival times are recorded at a laptop connected to the sniffer node. The deviation hence includes scatter from both the sniffer node and the laptop.

4.3 Case Study: Testbed Synchronization

Sensornet checkpointing captures the state of all nodes in a network. For synchronous checkpointing, all nodes must be checkpointed at the same network time. We freeze the entire network execution during time-consuming checkpointing operations. Hence, checkpointing synchronization error depends on when the freeze (and unfreeze) commands are received and handled by the different nodes.

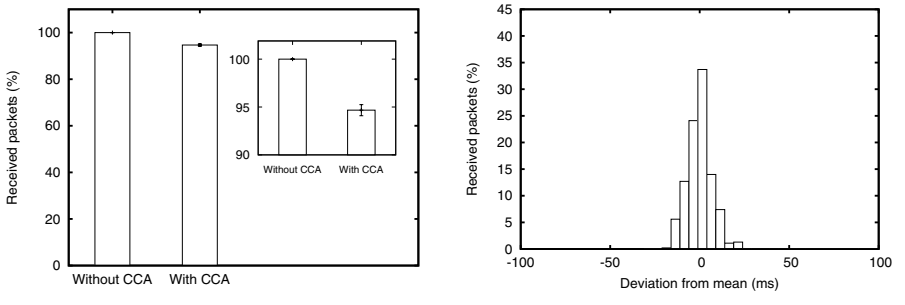


Fig. 6. Left: during the experiment without Clear-Channel Assessment (CCA), each node repeatedly transmitted the same radio packets to the sink. During the experiment with CCA, however, some packets were not retransmitted: the network behavior differed due to the radio surroundings. Right: the packet reception times, as measured by the sink, differs on the order of milliseconds showing that the pseudo-random network is repeated.

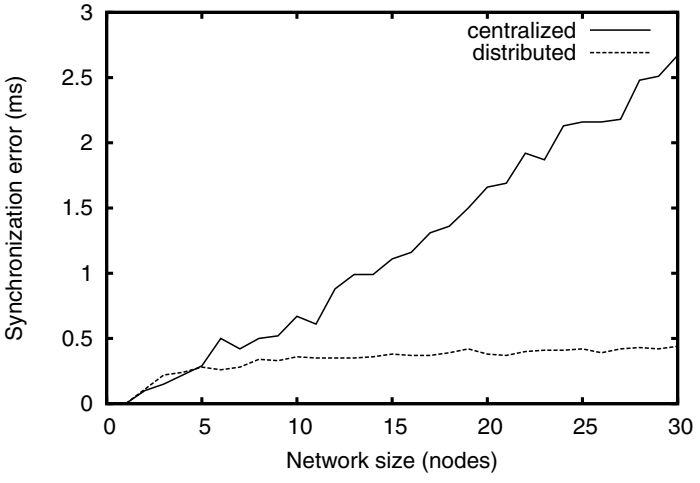


Fig. 7. Freezing and unfreezing nodes directly from subservers (distributed), as opposed to from the main server (centralized), significantly improves the synchronization: With 30 nodes distributed on 4 subservers, the synchronization error is less than 0.5 ms

In our implementation, checkpointing is performed via node serial ports. We reduce the network freeze synchronization error in two ways. First, node freeze commands are one-byte messages. Since freeze commands are sent out sequentially to each node, shorter commands results in better synchronization. Second, freeze commands originate close to the testbed nodes: at the subservers, minimizing message delivery time scatter.

We evaluate how network size affects checkpointing synchronization error. To measure synchronization error we use sensor node hardware timers. Each node dedicates a 32768Hz timer to the evaluation. The checkpointing software, connected to the testbed, continually freezes and unfreezes the testbed. Each time the network is frozen, the current hardware timer value of each node is extracted. Two consecutive node timer values are used to calculate the last test run duration: the network execution time as observed by the node itself.

We define the synchronization error of a test run as the maximum difference between all collected node durations. Hence, with perfect synchronization all nodes would report the same times during a test run, and the synchronization error would be zero. To avoid serialization effects, the order of which nodes are frozen and unfrozen each test run is random, as is the delay between each freeze and unfreeze command. We measure synchronization error on a testbed consisting of up to 30 nodes. The nodes are distributed on 4 subservers (10 + 9 + 8 + 3). See Figure 7 for the synchronization errors. In the distributed approach, the checkpointing software sends a single freeze command to each subserver, and the subservers forward the command to all connected nodes. In the centralized approach, the checkpointing software directly sends a command to each node.

The distributed approach is visibly more scalable and provides a significantly lower synchronization error than the centralized approach. With 30 testbed nodes the average synchronization error is 0.44 ms with the distributed approach, and 2.67 ms with the centralized approach.

5 Related Work

Distributed checkpointing and rollback is a standard technique for failure recovery in distributed systems [1,4] and is used e.g. in file systems [18], databases [20], and for playback of user actions [10]. Inspired by the substantial body of work in checkpointing, we use the technique to improve realism in sensor network simulation and repeatability in sensor network testbeds. To the best of our knowledge, we are the first to study distributed checkpointing with rollback in the context of wireless sensor networks.

Sensor network testbeds is a widely used tool for performing controlled experiments with sensor networks, both with stationary and mobile nodes [7,25]. Our work is orthogonal in that our technique can be applied to existing testbeds without modification. There is a body of work on sensor network simulators. TOSSIM [11] simulates the TinyOS sensor network operating system. Avrora [22] emulates AVR-based sensor node hardware.

A number of sensor network simulators allow a mixture of simulated nodes and testbed nodes. This technique is often called hybrid simulation [12,15,23,24]. Although hybrid simulation is designed to increase the realism by running parts of the network in a testbed, repeatability is affected negatively. Our work is originally based on the idea of hybrid simulation, but we identify key problems with hybrid simulation and show that synchronous checkpointing lessens the effects of these problems.

Several techniques for debugging and increasing visibility have been proposed for sensor networks. Sensornet checkpointing can be used as an underlying tool when implementing debugging and visibility, and can be combined with several of the existing tools.

NodeMD [9] uses checkpoints on individual nodes for detecting thread-level software faults. The notion of checkpoints is used differently in NodeMD: checkpoints are used by threads to signal correct execution. NodeMD furthermore stores logs in a circular buffer used to debug the node when an error has been detected. Sensornet checkpointing can be combined with many the features in NodeMD: a node-level error detected by NodeMD can trigger a network-wide checkpoint, and the circular buffers can be used as an efficient way to transfer messages between testbed and simulation.

6 Conclusions

We implement checkpointing for sensor networks. Our approach enables transferring network state between simulation and testbed. Several applications benefit from the approach, such as fault injection and testbed debugging. We show that

sensor network checkpointing enables repeatable testbed experiments and non-intrusive testbed execution details.

Acknowledgments

This work was partly financed by VINNOVA, the Swedish Agency for Innovation Systems. This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

References

1. Chandy, K., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3(1), 63–75 (1985)
2. Chipcon AS. CC2420 Datasheet (rev. 1.3) (2005)
3. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Workshop on Embedded Networked Sensors, Tampa, Florida, USA (November 2004)
4. Elnozahy, E., Alvisi, L., Wang, Y., Johnson, D.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)
5. Eriksson, J., Dunkels, A., Finne, N., Österlind, F., Voigt, T.: Mspsim – an extensible simulator for msp430-equipped sensor boards. In: Langendoen, K.G., Voigt, T. (eds.) EWSN 2007. LNCS, vol. 4373. Springer, Heidelberg (2007)
6. ASUSTek Computer Inc. (visited 2008-09-25), http://www.asus.com/
7. Johnson, D., Stack, T., Fish, R., Flickinger, D.M., Stoller, L., Ricci, R., Lepreau, J.: Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In: IEEE INFOCOM (2006)
8. Kotz, D., Newport, C., Gray, R.S., Liu, J., Yuan, Y., Elliott, C.: Experimental Evaluation of Wireless Simulation Assumptions. In: Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2004), October 2004, pp. 78–82 (2004)
9. Krunic, V., Trumpler, E., Han, R.: NodeMD: Diagnosing node-level faults in remote wireless sensor systems. In: MOBISYS 2007, San Juan, Puerto Rico (June 2007)
10. Laadan, O., Baratto, R., Phung, D., Potter, S., Nieh, J.: Dejaview: a personal virtual computer recorder. In: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, Stevenson, Washington, USA, pp. 279–292 (2007)
11. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the first international conference on Embedded networked sensor systems, Los Angeles, California, USA, pp. 126–137 (2003)
12. Lo, S., Ding, J., Hung, S., Tang, J., Tsai, W., Chung, Y.: SEMU: A Framework of Simulation Environment for Wireless Sensor Networks with Co-simulation Model. In: Cérin, C., Li, K.-C. (eds.) GPC 2007. LNCS, vol. 4459, pp. 672–677. Springer, Heidelberg (2007)
13. OpenWRT. OpenWRT Wireless Freedom (visited 2008-09-25), http://openwrt.org/

14. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with cooja. In: Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), Tampa, Florida, USA (November 2006)
15. Park, S., Savvides, A., Srivastava, M.B.: SensorSim: a simulation framework for sensor networks. In: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, pp. 104–111 (2000)
16. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling ultra-low power wireless research. In: Proc. IPSN/SPOTS 2005, Los Angeles, CA, USA (April 2005)
17. Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D.: Sympathy for the sensor network debugger. In: SenSys 2005: Proceedings of the 3rd international conference on Embedded networked sensor systems, San Diego, California, USA, pp. 255–267 (2005)
18. Rosenblum, M., Ousterhout, J.: The design and implementation of a log structured file system. In: SOSP 1991: Proceedings of the 13th ACM Symposium on Operating System Principes, Pacific Grove, Califonia, USA (1991)
19. ser2net application. Serial to Network Proxy (ser2net) (visited 2008-09-25), http://ser2net.sourceforge.net/
20. Son, S.H., Agrawala, A.K.: Distributed checkpointing for globally consistent states of databases. IEEE Transactions on Software Engineering 15(10), 1157–1167 (1989)
21. Takai, M., Martin, J., Bagrodia, R.: Effects of Wireless Physical Layer Modeling in Mobile Ad Hoc Networks. In: Proceedings of MobiHoc 2001 (October 2001)
22. Titzer, B.L., Lee, D.K., Palsberg, J.: Avrora: scalable sensor network simulation with precise timing. In: Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN) (April 2005)
23. Watson, D., Nesterenko, M.: Mule: Hybrid Simulator for Testing and Debugging Wireless Sensor Networks. In: Workshop on Sensor and Actor Network Protocols and Applications (2004)
24. Wen, Y., Wolski, R.: Simulation-based augmented reality for sensor network development. In: Proceedings of the 5th international conference on Embedded networked sensor systems, pp. 275–288 (2007)
25. Werner-Allen, G., Swieskowski, P., Welsh, M.: MoteLab: a wireless sensor network testbed. In: Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005, pp. 483–488 (2005)
26. Woehrle, M., Plessl, C., Beutel, J., Thiele, L.: Increasing the reliability of wireless sensor networks with a distributed testing framework. In: EmNets 2007: Proceedings of the 4th workshop on Embedded networked sensors, pp. 93–97. ACM Press, New York (2007)
27. Yang, J., Soffa, M.L., Selavo, L., Whitehouse, K.: Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In: Proceedings of the 5th international conference on Embedded networked sensor systems, pp. 189–203 (2007)

SRCP: Simple Remote Control for Perpetual High-Power Sensor Networks

Navin Sharma, Jeremy Gummeson, David Irwin, and Prashant Shenoy

University of Massachusetts, Amherst
Department of Computer Science
{nksharma,gummeson,irwin,shenoy}@cs.umass.edu

Abstract. Remote management is essential for wireless sensor networks (WSNs) designed to run perpetually using harvested energy. A natural division of function for managing WSNs is to employ both an in-band data plane to sense, store, process, and forward data, and an out-of-band management plane to remotely control each node and its sensors. This paper presents *SRCP*, a Simple Remote Control Protocol that forms the core of an out-of-band management plane for WSNs. SRCP is motivated by our target environment: a perpetual deployment of high-power, aggressively duty-cycled nodes capable of handling high-bandwidth sensor data from multiple sensors. The protocol runs on low-power always-on control processors using harvested energy, distills an essential set of primitives, and uses them to control a suite of existing management functions on more powerful main nodes. We demonstrate SRCP's utility by presenting a case study that (i) uses it to control a broad spectrum of management functions and (ii) quantifies its efficacy and performance.

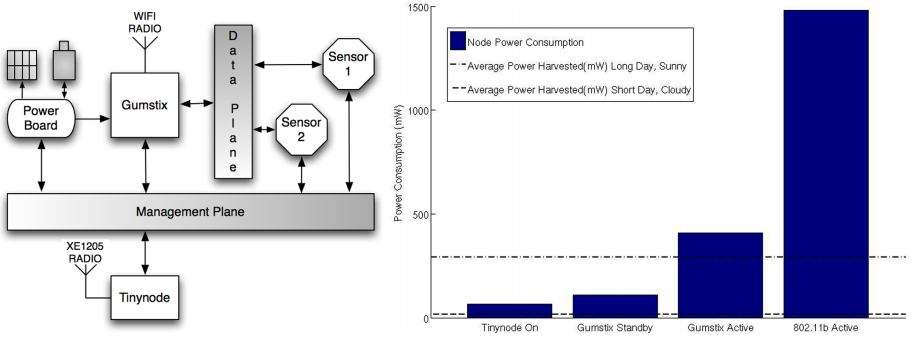
1 Introduction

Perpetual wireless sensor networks (WSNs) consist of nodes that harvest environmental energy (*e.g.*, solar, wind, thermal, vibration) to indefinitely sustain data collection, storage, processing, and transmission without physical interaction[1]. Perpetual operation is ideal for WSNs composed of nodes that are time-consuming to construct and deploy. However, the long lifetime of perpetual WSNs elevates the importance of remote management functionality. Effective remote management has three characteristics:

– **Visibility.** A sensor node is visible if its state *is known or can be queried*. High visibility permits simple, direct monitoring of each node with high accuracy, while low visibility forces complex, indirect monitoring based on collected sensor data, past node states, or neighbor states. Visibility is essential

[1] This research was supported in part by an UMass President's Science and Technology award and NSF grants CNS-0626873, CNS-0615075, CNS-0520729, and EEC-0313747. We also acknowledge Deepak Ganesan for providing us feedback on early versions of this work.

U. Roedig and C.J. Sreenan (Eds.): EWSN 2009, LNCS 5432, pp. 358–374, 2009.
© Springer-Verlag Berlin Heidelberg 2009



(a) Management Plane Overview (b) Power Consumption

Fig. 1. Our prototype (a) uses a management plane. Power states are shown in (b) from a typical sunny summer day and cloudy winter day.

for discovering the software bugs or hardware faults that impair WSN operation; recent work proposes elevating low-cost visibility to a fundamental WSN design principle [21].

- **Accessibility.** A sensor node is accessible if its state is known or can be queried, and *can also be altered*. High accessibility permits simple, direct remote node maintenance, including altering application-level software, kernel-level software, and hardware states (*e.g.*, power states). Low accessibility increases the scope of problems that require physical access to a node.
- **Interactivity.** A sensor node is interactive if its state *is both visible and accessible with tolerable latency*. High interactivity permits a controller to react to visible changes in the WSN by accessing a node and changing its state, and quickens the upgrade-test-debug development cycle required to produce robust software. Low interactivity limits the WSN's capability to adapt its operation to unexpected operational or environmental changes, and slows or impairs the software development cycle.

Visibility, accessibility, and interactivity are highly correlated with a node's duty cycle, since a controller is unable to query or alter state while a node is powered down. As a result, simultaneously satisfying all three characteristics is challenging for resource-constrained WSNs. Previous approaches primarily address only a single aspect of remote management using *in-band* techniques that share a single wireless channel and node processor between both management-centric and data-centric tasks [11,18,20,21,23,24]. In-band approaches are *invasive*: they consume limited resources that interfere with the primary tasks of sensing, storing, processing, and transmitting data.

Out-of-band management isolates management tasks on a separate always-on control processor and radio attached to each node. The approach divides WSN functions between a *data plane* that senses, stores, processes, and transmits data, and a *management plane* that ensures continuous visibility, accessibility, and interactivity. The division takes advantage of the natural distinction between

control traffic (short infrequent interaction) and data traffic (bulk data transfer). Out-of-band management is also a common technique for diagnosing and repairing node failures in other distributed systems, such as networks and data centers. While the energy costs of using a separate per-node control processor and radio preclude out-of-band management in some scenarios, the approach is well-suited for *high-power* WSNs that handle data from one or more high-bandwidth sensor streams and engage in computationally-intensive processing.

Examples of such high-power sensor applications include networks of weather sensing radars [14] and camera networks [19]. Our target application is monitoring river ecologies using a diverse array of connected sensors, including underwater camera, hydrophone, water quality, geologic imaging, and temperature sensors. Since high-power WSNs already require enough harvested energy to support a powerful node platform (*e.g.*, an iMote2, Gumstix, or embedded PC-class node), it is feasible to continuously operate a less powerful control processor and radio that uses a small fraction of the main node's power (*e.g.*, a TinyNode, TelosB, or MicaZ mote) and has a minimal impact on the data plane's operation.

To illustrate, Figure 1(a) provides an overview of our prototype's management plane, which uses a Tinynode with an XE1205 radio as the control processor and a Gumstix [1] with a PXA-based microcontroller and commodity 802.11b WiFi radio as the high-power main node. The XE1205's long range (1.43 miles at 4.8kbps [6]) makes it particularly attractive for out-of-band management. Importantly, the Tinynode control processor is also able to control the sensors directly without consuming additional energy by powering the main node. Figure 1(b) shows average energy production from an attached 4"x8" solar panel on a typical sunny summer day and cloudy winter day compared with the prototype's average energy consumption in different power states. The measurements demonstrate that even on a worst-case cloudy winter day the control processor is able to remain on continuously using a small amount of buffered energy from a battery, while the main node must remain mostly off due to the high energy cost of operating its processor, radio, and sensors.

Contributions. This paper presents *SRCP*, a Simple Remote Control Protocol for use on low-power control processors and radios that forms the core of a non-invasive out-of-band management plane for perpetual high-power WSNs. A key design principle of SRCP is to expose a narrow set of management primitives *without* defining management services; as a result, much of its power derives from connecting controllers to existing management functions provided by high-power hardware platforms and their software. SRCP's narrow set of primitives is able to unify a broad spectrum of management functions. In particular, we show that the protocol is able to monitor the health of the network at fine granularities (1 update every 250 milliseconds for a 5 hop network), support interactive debugging sessions using a low-bandwidth radio (less than 2 second latency per directive), and non-invasively transfer bulk software updates using a DTN routing protocol.

2 Related Work

In-band Management. In-band techniques improve visibility, accessibility, and interactivity within the confines of a node's energy constraints and duty cycle, and are orthogonal to SRCP, which assumes an always-on control processor and radio.

Sympathy [18] and PCP [21] improve visibility indirectly by correlating lack of sensor data from a node with failure and then diagnosing the root cause by traversing a decision tree of likely possibilities. NodeMD addresses visibility, by using a runtime detection system to detect faults, and interactivity, by catching many faults before they disable the system and enabling a debug mode [11]. Clairvoyant [24] and Marionette [23] also improve interactivity by enabling interactive debugging. In contrast, SRCP is able to connect operators to existing debuggers available for commodity OSes, such as GDB, to enable interactive debugging of data plane software.

Dissemination techniques for efficiently updating node software, including Trickle [13] and Deluge [10], improve accessibility. Since we target high-power WSNs using complete network stacks for SRCP, more general communication techniques, such as end-to-end TCP connections or DTN-style store-and-forward paradigms, are possible for both disseminating data plane software updates and transmitting sensor data. Finally, SNMS recognizes the importance of separating the management plane from the data plane by decoupling them to the extent possible using an in-band approach and implementing a broad set of management services [20]; SRCP completely decouples the two planes and provides narrow primitives for connecting to existing management services.

Out-of-band Management. Out-of-band management is a necessity in sensor testbeds that support multiple experiments over time, as in MoteLab [22] or Trio [7]. These testbeds utilize a back-channel or control processor to provide continuous visibility and access to nodes. However, the main purpose of the back-channel is to enforce the testbed's scheduling policies, ensuring that no experiment monopolizes testbed resources, and to deploy experiment-specific software. Perhaps most related to SRCP is the Deployment Support Network (DSN), which mitigates the need for a wired back-channel in testbeds by attaching a separate battery-powered control processor and radio, called a DSN-node, to each main testbed node [8].

In contrast to DSN, our target application is a perpetual deployment, which warrants a focus on a simple protocol suitable for low-power control processors and radios that adaptations to DSN-node software do not easily address. DSN-nodes implement common testbed services, such as event logging, interactive debugging, and software distribution, whereas SRCP is protocol-centric and simply enables remote access to software services and low-level hardware functions that already exist for high-power nodes. The choice of radio for DSN highlights the different focus: DSN-nodes use Bluetooth for their backbone wireless network while we choose the XE1205. The XE1205 is a low-power, long-range, and low-bandwidth radio that allows our management plane to operate over longer

distances than short-range Bluetooth radios, decreasing both the incidence of network partitions due to control processor failure—since it may be possible to "hop over" failed nodes—and the available bandwidth for the management plane. The XE1205 is also not invasive: it minimally impacts node energy consumption and eliminates traffic conflicts with a shorter range main node radio.

While SRCP does assume dual-radio/dual-processor nodes, previous work on these systems focuses primarily on dynamically assigning tasks to components to adjust the energy/performance ratio on specific hardware platforms [3,5,16]. SRCP's focus is more narrow but also more general: it defines a simple protocol that is a foundation for remote management of a broad range of hardware platforms with control processors. In particular, Leap is a hardware/software architecture using two battery-powered processors and radios that shares our focus on high-power WSNs [16]. Leap and SRCP share a common goal but a different focus: SRCP is a simple, extensible, and hardware-independent protocol that distills a small set of core remote management functions for always-on control processors, while Leap combines a hardware platform that supports fine-grained energy monitoring with algorithms for dynamically scheduling tasks to processors to minimize energy consumption.

3 SRCP: A Simple Remote Control Protocol

SRCP is inspired by SNMP, a standard for managing network-attached devices. SNMP's primary use is *monitoring* wired network devices, while SRCP's primary use is monitoring and *controlling* wireless network devices using an external control processor. Figure 2 shows an overview of SRCP's node architecture and software artifacts. Each control processor runs an instance of an SRCP *agent* that implements the protocol. A *slave agent* also runs on each main node and interacts with its agent to permit interaction with the main node's software services.

SRCP is a basis for both out-of-network and in-network control. With out-of-network control, an SRCP *controller* injects protocol messages into the management plane from a well-connected base station within range of at least one agent. For example, the controller may disseminate protocol messages that power down all main nodes during uninteresting periods (*e.g.*, at night in a camera network), or power up main nodes on-demand during interesting periods of plentiful energy (*e.g.*, to apply synchronized software updates or initiate immediate sensor data collection and transmission).

With in-network control, one agent issues protocol messages to another agent to control its behavior. For example, an agent for a node with a full buffer of sensor data could issue a control message to activate an upstream node in order to transmit the data and free buffer space for additional sensing. The use of SRCP for in-network control requires inter-node cooperation to ensure that the decisions made by agents using their local knowledge result in acceptable global WSN-wide outcomes that efficiently use energy, network, processing, and storage resources. While the protocol does support in-network control, techniques for

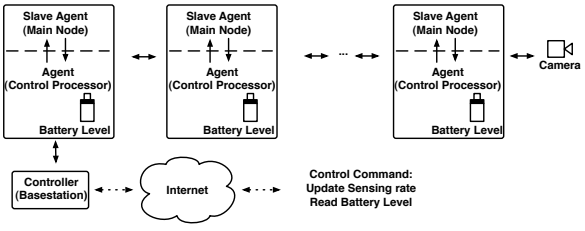


Fig. 2. An SRCP agent runs on each control processor and a slave agent runs on each main node

inter-node cooperation are outside the scope of this paper; instead, we focus on out-of-network control using a controller running at a base station.

3.1 Protocol

The protocol is based on short *control messages*, which may be fragmented into one or more transmitted *packets*. Each message is sent by an SRCP controller to one and only one agent, and directs the agent to take a specific *action* at a node that produces an *outcome*. The one-to-one communication paradigm is simple and enables management of each node as a distinct entity, rather than in-band approaches that expose WSNs as a single aggregate. Upon message receipt, the agent invokes the specified action and then transmits a *response message* to the controller that encapsulates an acknowledgement of control message receipt, an indication of the action's success or failure, and an action-specific payload that encodes a description of its outcome.

A key goal of SRCP is to separate remote management primitives from specific remote management services; the primitives are general enough to serve as a foundation for monitoring any hardware platform or controlling any software on the main node. As a result, SRCP is extensible since some actions require hardware or software support from the main node or attached sensors that may vary across platforms. Developers register hardware-related actions (Execution and State) prior to deployment by defining a unique index number and linking the action's logic to the implementation at compile time, while controllers are able to register other actions (Conditional and Connection) *in situ* post-deployment.

3.2 Primitives

The protocol distills actions into four fundamental classes of remote management primitives: Executions, States, Conditionals, and Connections. Each class consists of one or more distinct actions, where the controller and agent associate each action with a unique integer index. Control messages include this index, which also identifies the message class, as part of the message payload to direct the agent to act on a specific Execution, State, Conditional, or Connection.

Executions. An execution is an action that affects the operational state of the main node or any attached sensors. In particular, executions make visible hardware/software control of the main node and attached sensors that would otherwise not be available when the main node is inactive and powered down. For instance, our reference implementation includes an execution action that directs the slave agent to invoke an arbitrary process on the main node and return its standard output and standard error in a response message. If the main node OS supports fine-grained resource control (*e.g.*, Resource Containers [4] or PixieOS tickets [15]), then execution actions may also dynamically control node energy, CPU, memory, or bandwidth usage. Other examples include:

– Power-on Main Node; Power-off Main Node; Sleep Main Node (ACPI S3); Hibernate Main Node (ACPI S4); Power-on Main Radio; Low-power Main Radio; Power-off Main Radio; Main Node Process Execution; Main Node File Transfer; Take Picture; Transfer Picture to Flash; Main Node Alive Ping; Reboot Main Node Standard Kernel; Reboot Main Node Clean Kernel;

The payload of the control message includes its index along with execution-specific data, while the payload of the response message includes details of the execution's outcome.

States. Actions may read, and in some cases write, state variables stored by the control processor using its limited on-board memory. SRCP divides state variables into two categories: environmental states and management states. The value of environmental state, such as the current battery level, is dictated by the environment and is read-only, while the value of management state, such as a routing table entry, is writeable remotely using a control message. Examples of environmental and management states include:

– **Environmental States.** Battery Energy Level; Solar Power Production; Platform Power Consumption; Main Node Reboot Counter; Control Processor Reboot Counter; Current Time;
– **Management States.** Voltage Level; Flash Memory (via JTAG); Main Node Processor Registers (via JTAG); Routing Table Entry; Conditional Period; Environmental State Update Period;

The agent automatically monitors and updates the value of environmental states at a predefined time granularity. The payload of a state-based control message includes its index, a flag indicating a read or write operation, and a value if applicable. The response message payload includes the value of the state and any acknowledgements.

Conditionals. In some cases, it is necessary for an agent to react immediately to local conditions or at a prespecified time. A Conditional action invokes an execution or state-based action based on a condition. The condition is a boolean expression composed of environmental or management states and boolean operators. The protocol supports two conditional types: one-time and continuous.

A one-time Conditional action executes a single time when a condition is true, while a continuous Conditional action executes every time a condition is true every configurable time period. The controller is able to use control messages to dynamically add, remove, or modify an action's condition. The payload of a Conditional control message includes an index, a flag indicating whether to set a new condition or delete an existing one, an Execution or State action to execute, and the condition. The response message simply acknowledges the action's success or failure. Note that once a conditional is set, the controller will receive asynchronous response messages associated with its Execution or State actions.

Connections. Long-lived interactive sessions between a controller and a main node require reliable end-to-end communication not possible using short control messages. To accommodate interactive sessions, the agent forwards packets marked as connection actions directly to its slave agent. These packets are opaque to the agent and are only interpreted by the slave agent; the intent is to support network layer tunneling and end-to-end connections between the controllers and slave agents, which may both implement complete network stacks.The payload of a connection packet includes its index along with opaque data interpreted by the slave agent (*e.g.*, TCP packets). The payload of the response message includes its index along with opaque data interpreted by the controller (*e.g.*, TCP acknowledgements).

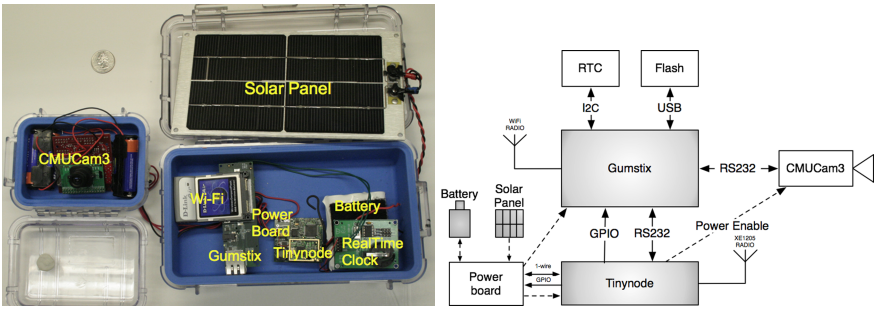
4 Implementation

We have written a reference implementation of SRCP [2]. While the protocol is platform-independent, the reference implementation is intended for a mote-class control processor with a Linux-capable main node and supports an agent for TinyOS and a slave agent for Linux. The implementation utilizes hardware features (*e.g.*, JTAG, numerous power states) that require a compatible hardware platform, although its core functions are portable to other platforms utilizing a TinyOS-based control processor and Linux-based main node.

4.1 Hardware Prototype

Since many of the protocol's functions interact with specific hardware features of the main node and sensors, we built a hardware prototype (Figure 3(a)) that fully utilizes it. The prototype is a general-purpose node platform for high-power WSNs; it uses a Tinynode control processor with a low-power MSP430 microcontroller, 512Kb of on-board flash, and an XE1205 radio. The XE1205 radio is attractive for out-of-band management since it does not interfere with the data plane's 802.11b radio, and is able to trade bandwidth for range. The measured range of the radio has been shown to be 2.3 kilometeres (1.43 miles) at a bandwidth of 4.8kbps, exceeding the range of the Mica2 or Telos by at least a factor of 4 [6]. Additionally, we also evaluated the implementation on TelosB motes using the more capable CC2420 radio.

[2] http://lass.cs.umass.edu/



(a) Hardware Prototype (b) Block Diagram

Fig. 3. A photograph (a) and a block diagram (b) of our hardware prototype

The prototype uses a Gumstix with a PXA-based microcontroller and a commodity 802.11b WiFi radio for the main node. The Gumstix runs a instance of Linux that supports standard Linux utilities. We attach the CMUCam3 imaging sensor as a representative example of a high-bandwidth sensor [19]. The prototype also includes external flash for additional Gumstix storage and a real-time clock that the Gumstix and Tinynode use to periodically synchronize their notion of time. Figure 3(b) shows a block diagram of the prototype. Communication between the Tinynode's agent and the Gumstix's slave agent occurs over a serial RS-232 connection. The main powerboard regulates charging from energy produced by a SPE-350-6 SolarWorld solar panel, stores it in 3.7V Ultralife rechargeable battery with a capacity of 6.1 Amp-hours, and distributes it to the Tinynode, Gumstix, and CMUCam3 sensor. The materials for each prototype node cost approximately $650.

To increase management flexibility, the Tinynode is capable of independently controlling functions on the Gumstix, its WiFi radio, and the CMUCam3 sensor. A modular hardware platform, such as our prototype or LEAP [16], is a useful paradigm for high-power WSNs, since they allow a controller to independently power and operate each component. Our experiences demonstrate that SRCP is flexible enough to support a range of functions on different devices in such a platform.

4.2 Software Prototype

The SRCP reference implementation, described below, includes an agent written in NesC for TinyOS, a slave agent written in C for Linux, and a simple controller for a base station written in C.

– **Agent.** The agent implements the protocol from Section 3 and supports the example Execution and State actions from that section. We discuss details of network communication in our prototype (*e.g.*, routing, packet format) in Section 4.3.

- **Slave Agent**. The slave agent runs on Linux and integrates with the TinyOS serial forwarder to send and receive control messages from the agent. In addition to interpreting Connection messages, the slave agent includes support for specific Execution actions that integrate with software supported by the node. For instance, our implementation includes support for executing processes and receiving standard error and out using control messages, and direct file transmission over the management plane. Both actions are suitable for short-lived interactions (*e.g.*, quick process execution or small files), or as a "last resort" for connecting to the main nodes when communication via WiFi is impossible. As discussed in Section 5, our slave agent also incorporates a DTN routing reference implementation for disseminating bulk software updates and gathering sensor data.
- **Controller.** The controller includes a management shell and GUI dashboard to manually inject control messages and view their responses. Designing policies that dynamically adjust the WSN's behavior based on both environmental conditions and data requirements is the subject of future work. The base station includes both an 802.11b WiFi radio for data plane communication and a root Tinynode with an XE1205 radio running an agent connected over an RS-232 serial link. The controller integrates with the TinyOS serial forwarder to forward control messages over the serial link to a "first-hop" SRCP agent that routes them to their destination using the XE1205 radio.

IP Tunneling. To support TCP/IP flows between a controller and slave agent using Connection messages, both include a Control Message/IP proxy for establishing IP tunnels. The proxy (i) captures egress IP packets, fragments them into control or response messages, and forwards them to its agent for transmission over the management plane and (ii) reassembles ingress control messages into IP packets and injects them into the network stack.

The proxy currently utilizes the Linux Netfilter library for capturing egress packets from the network stack and Linux raw sockets for injecting ingress packets back into the network stack, although we are exploring the benefits of TUN/TAP-based implementation. We assign the base station and Gumstix nodes an IP address in the 172.16.0.0/16 subnet; the proxy then uses standard `iptables` rules to capture all packets destined for the subnet for tunneling. While SRCP could use the an implementation of 6LoWPAN to forward IPv6 packets without tunneling them, the XE1205 radio does not support 6LoWPAN's standard 127 byte packet size [17]. 6LoWPAN is not necessary for our prototype since the control processor does not serve as a connection end-point, and the Gumstix main node supports a complete network stack.

Since long-range radios cannot sustain high bit rates, the proxy includes an implementation of Van Jacobson header compression (IPcomp) from RFC 1144 to reduce the length of TCP/IP from 40 bytes to 1 or 2 bytes on average. IPcomp is useful for reducing overhead in interactive sessions composed of a series of small packet transmissions (*e.g.*, ASCII characters), where TCP/IP headers can consume up to 50% of a TCP packet's size. Section 5 quantifies the effect of IPcomp on interactivity. Additionally, we used suggestions from RFC 3150 to

set TCP parameters for low speed and unreliable lengths (*e.g.*, lowering TCP Maximum Segment Size (MSS) from its default of 1500 bytes).

4.3 Management Plane Communication

A radio for out-of-band management values transmission range and energy efficiency over bandwidth, since disconnected nodes impede visibility, accessibility, and interactivity. The XE1205 radio we choose for our prototype imposes limitations on the maximum possible packet size: the radio does not reliably support packets larger than 28 bytes (the default AM packet size in TinyOS) due to a 16 byte send/receive FIFO buffer that requires 50 μsec to empty[3]. As a result, our implementation imposes limits on header sizes and does not provide a reliable transport protocol, as discussed below.

Packet Format. Each control message packet includes a minimal header with fields for a message identifier, a sender's identifier, a destination's identifier, a fragment number, a message length, and a time-to-live value. Each identifier uniquely identifies the message, sender, and destination; the destination uses the fragment number and message length to reassemble the message; the time-to-live value defaults to the network's diameter and ensures that the network eventually drops packets that cannot reach their destination. We use 2 bytes for the message length field and a single byte for the remaining fields. Thus, our implementation uses 7 bytes out of each 28 byte packet for headers (25%) and 21 bytes for the payload (75%). The sender identifier, destination identifier, and time-to-live fields must increase to scale the protocol to networks larger than 255 nodes or with a diameter greater than 255.

Reliable Communication. While agents acknowledge messages using responses, as described in Section 3.1, the controller and agent do not acknowledge packets end-to-end, since acknowledgements at each level of the network stack consume bandwidth. The implementation uses only simple link-layer acknowledgments provided by TinyOS's AM abstraction to ensure reliable per-hop packet transmission. An agent retransmits each packet if it does not receive a link-layer acknowledgement within timeout t, and after k retransmissions it drops the packet. Without end-to-end packet acknowledgements, the loss of a single packet prevents the delivery of an entire control or response message. As a result, the implementation limits many control and response messages to a single packet, although the size of some messages, such as an encoded outcome in a response message, may be an arbitrary length.

Routing. Finally, agents must be able to route packets from the controller to the packet's destination. Our implementation assigns each agent a simple static identifier, and agents forward any received packet with a destination identifier that does not match its own to the next hop. Each agent maintains a routing table as special management states. To determine the next hop, the agent looks up the destination identifier in its set of management states, and interprets the

[3] The time taken to empty the buffer 3 times per single packet reception or transmission results in unacceptable loss rates.

value as the next hop identifier. The controller is able to alter routes dynamically using State actions, although we have not explored dynamic approaches to routing.

5 Evaluation

We evaluate SRCP with a case study that exemplifies the actions we envision a WSN controller using to manage a network, and quantifies SRCP's performance along three axis: visibility, interactivity, and accessibility.

Visibility. Wachs et al. define a visibility metric as the energy required to diagnose a failure [21]. Table 1 reports microbenchmarks of the CPU time and energy cost to a Tinynode in our prototype to execute an action from each class, and demonstrates that individual actions are non-invasive and impose little energy cost on the data plane. The primary energy cost derives from keeping the control processor active (see Figure 1(b)) and not from executing individual actions.

Since enough energy exists to continuously operate SRCP's control processor and radio, an energy-centric visibility metric is not appropriate. Instead, SRCP's primary constraint is management plane network bandwidth; as a result, we define visibility as the rate (messages/second) at which a controller is able to observe changes in the state of the network. The frequency of node health updates is a direct measure of visibility since they expedite the discovery of anomalies or failures.

In Figure 4(a), we observe the loss rate for node health updates as a function of their send rate. For the experiment, we use a configuration of 5 nodes in a simple chain topology, where each node is 15 meters from its neighbors. Each node uses a Conditional action that reports battery level and pings the main node

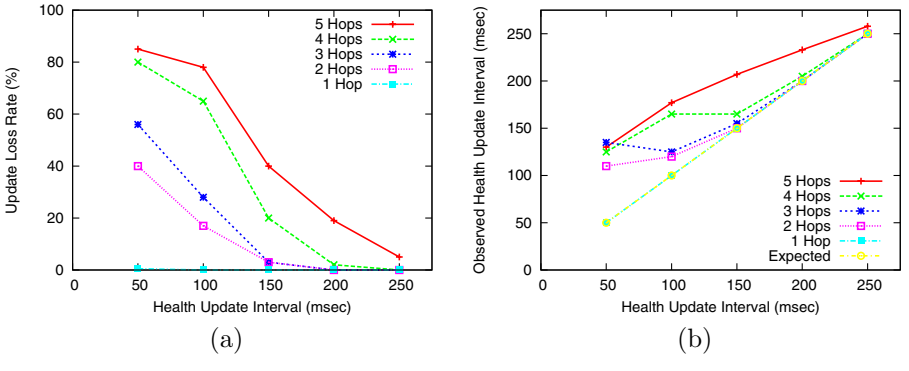


Fig. 4. Measurements of the percentage of update losses (a) using Conditional actions to monitor node health over a range of intervals for different network sizes. The observed health update interval (b) is the interval between health updates seen at the controller.

Table 1. *Max* shows the maximum number of times the Tinynode command could be executed based on our prototype's battery with capacity 81,360 joules

Primitive	*Command*	*Power (µjoules)*	*Max*
Execution	Wakeup Gumstix	0.551	$1.47x10^{11}$
Set Conditional	Wakeup Gumstix in 5 minutes	0.580	$1.40x10^{11}$
Read State	Sensing rate	13.92	$5.84x10^{9}$
Write State	Sensing rate	13.97	$5.82x10^{9}$
Connection	Transmit 28 byte packet	0.560	$1.45x10^{11}$

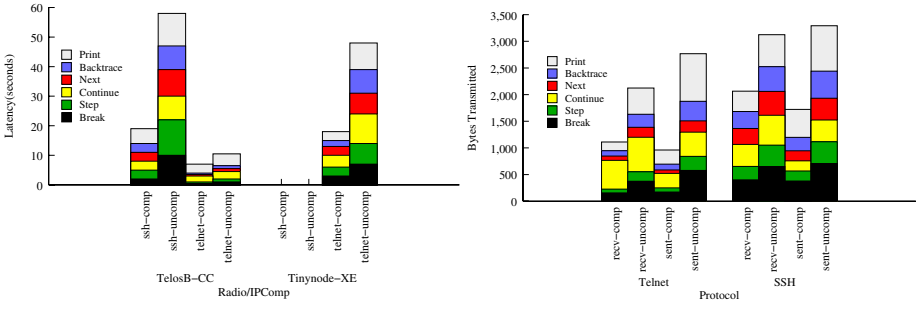
for liveness; note that the health updates could include any of the management or environmental states listed in Section 4. The x-axis shows the health update interval for each node and the y-axis shows the percentage of updates lost in the management plane and not delivered to the controller. The result demonstrates that the prototype is able to sustain an update interval of 250 milliseconds in a 5 hop network without experiencing significant losses due to network congestion, allowing a controller to detect any node anomalies (*e.g.*, low battery level, failed main node) at a 250 millisecond granularity. In Figure 4(b), we plot, for the same experiment, the observed average health update interval seen at the controller, demonstrating that the observed rate is close to the expected interval even when experiencing congestion-related losses.

Our results indicate that the management plane should be able to monitor a network of N nodes at an interval of $250N/5 == 50N$ milliseconds; for a network of 100 nodes this translates to an update interval of 5 seconds. In practice, we expect the controller to observe the entire WSN at a coarse granularity and focus in on specific regions with a finer granularity once an anomaly is detected.

Interactivity. Operators must diagnose and repair problems in the data plane that impair operation. Rather than indirectly diagnosing a problem, as in Sympathy [18], SRCP uses Connection actions to enable interactive debugging sessions on the main node. As with visibility, the primary constraint is network bandwidth. Figure 5(a) measures the latency for a representative interactive GDB session using a set of common debugging commands over both Telnet and SSH with and without IPcomp in a 5 hop network [4]. Since the XE1205 radio prevents packet sizes greater than 28 bytes, we also show results using TelosB motes with a CC2420 radio to study the impact of larger packet sizes on latency.

The measurements show that interactive sessions using Telnet and IPcomp are possible for both radios. However, the XE1205's 28 byte packet size limitation prevents tolerable interactive sessions for SSH with or without IPcomp. IPcomp has a significant impact on both Telnet and SSH, improving latency by at least a factor of 2 for all commands. Figure 5(b) shows total bytes sent and received

[4] The communication cost of the remote GDB protocol, which transfers individual assembly instructions, consumes enough network bandwidth to prevent tolerable interactive sessions.



(a) Latency for GDB (b) Bytes Sent/Received for GDB

Fig. 5. Telnet sessions using IPcomp and 100 byte packets on the CC2420 radio perform best for interactive GDB debugging sessions (a). IPcomp has a factor of 2 impact on total bytes transmitted (b).

for both sessions. In the best case—Telnet with IPcomp—the interactive latency is less than 3 seconds for each command for the XE1205 and less than a second for the TelosB. Figure 6(a) shows that the session latency increases modestly with the number of network hops; extrapolating the trend indicates that a 30 hop network path should experience sub-10sec latency for the total session with the XE1205 using Telnet/IPcomp.

Accessibility. Once an operator diagnoses a problem using an interactive debugging session it may be necessary to update the node's software. SRCP enables accessibility at each level of a node's hardware/software stack. At the lowest level, our implementation interacts with the Gumstix JTAG controller to provide remote access and control of its hardware, as proposed in [9].

The SRCP agent uses an execution action to expose remote access to JTAG through a set of 4 GPIO pins; in a conventional setting, these GPIO pins would connect to a PC through a USB or parallel-port JTAG connector. JTAG integration enables two capabilities: (i) direct control to read and write processor registers, including the instruction register, and clock the CPU and (ii) direct reading and writing of Flash memory. The first capability is useful for running hardware diagnostics on nodes without an operational OS, while the second capability is useful as a "last resort" for reading the state of flash off a failed node or writing flash directly to reconfigure a failed node from scratch (*e.g.*, install a new bootloader/minimal kernel). Our microbenchmarks show that it takes 205.3 seconds to write the Gumstix's 163kB uBoot bootloader to Flash one word at a time in a single hop network.

At the next level of the stack, our implementation integrates with the Gumstix's uBoot bootloader to implement a SafeBoot mechanism as an Execution action. SafeBoot allows the controller to select either a "safe" kernel or a standard kernel when rebooting a node. The safe kernel is preloaded on Flash and is read-only, while the controller may update and modify the standard kernel to upgrade drivers or propagate patches. To initiate SafeBoot, the SRCP agent sets

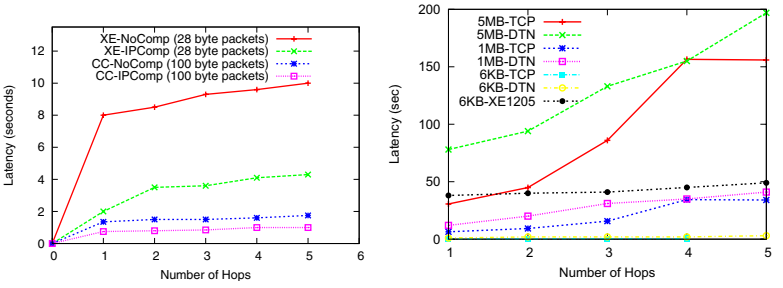


Fig. 6. Latency for interactive sessions increases modestly with the number of network hops (a). We use DTN (b) to opportunistically and non-invasively transfer bulk software updates in the data plane.

a GPIO pin on the Gumstix prior to applying power; the mechanism requires a minor modification to uBoot to check the pin state prior to boot to determine the appropriate kernel. A controller may also use the SafeBoot mechanism or the reboot mechanism in conjunction with Conditional actions to implement watchdog or grenade timers that periodically bring nodes to a clean state.

At application-level, our slave agent incorporates the reference implementation of DTN2 for non-invasive bulk software updates [2]. Software updates represent the one area where the management plane, due to bandwidth limitations, leverages the data plane for tasks that are not data-centric. Rather than requiring the controller to coordinate activation of every upstream node to update a downstream node's software using direct TCP connections, which would impact the operation of the data plane, we use DTN to opportunistically route data as main nodes become active. Figure 6(b) compares the latency to transfer different size files over DTN and TCP in the data plane and using an SRCP Execution action in the management plane. The benchmark demonstrates that the management plane is not suitable for software updates or other bulk data transfers (6kb takes 38sec over a single hop), and that, while not performing as well as TCP, DTN is a useful tool for non-invasive bulk data transfer over the management plane (5MB takes 200sec over 5 hops).

6 Conclusion

The energy demands of emerging high-power WSNs permit non-invasive out-of-band management through an always-on control processor powered by harvested energy. SRCP enables the paradigm using agents to monitor or change a node's operational, environmental, and management state and connect to its software services. Our evaluation shows that SRCP's primitives unify a broad range of management functions, and its performance is acceptable and non-invasive.

References

1. Gumstix (accessed, February 2008), http://www.gumstix.com
2. Delay Tolerant Networking Research Group (accessed, February 2008), http://www.dtnrg.org/wiki/Code
3. Bahl, P., Adya, A., Padhye, J., Walman, A.: Reconsidering Wireless Systems with Multiple Radios. SIGCOMM Computer Communications Review 34(5), 39–46 (2004)
4. Banga, G., Druschel, P., Mogul, J.C.: Resource Containers: A New Facility for Resource Management in Server Systems. In: Proceedings of the Symposium on Operating System Design and Implementation, February 1999, pp. 45–58. New Orleans, Louisiana (1999)
5. Draves, R., Padhye, J., Zill, B.: Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks. In: Proceedings of the International Conference on Mobile Computing and Networking, September 2004, pp. 114–128 (2004)
6. Dubois-Ferriére, H., Fabre, L., Meier, R., Metrailler, P.: TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications. In: Proceedings of the International Conference on Information Processing in Sensor Networks, Nashville, Tennessee, April 2006, pp. 358–365 (2006)
7. Dutta, P., Hui, J., Jeong, J., Kim, S., Sharp, C., Taneja, J., Tolle, G., Whitehouse, K., Culler, D.: Trio: Enabling Sustainable and Scalable Outdoor Wireless Sensor Network Deployments. In: Proceedings of the International Conference on Information Processing in Sensor Networks Nashville, Tennessee, April 2006, pp. 407–415 (2006)
8. Dyer, M., Beutel, J., Kalt, T., Oehen, P., Thiele, L., Martin, K., Blum, P.: Deployment Support Network - a Toolkit for the Development of WSNs. In: Proceedings of the European Conference on Wireless Sensor Networks (January 2007)
9. Eberle, H., Wander, A., Gura, N.: Testing Systems Wirelessly. In: Proceedings of the IEEE VLSI Test Symposium, pages 335 (April 2004)
10. Hui, J.W., Culler, D.: The Dynamic Behavior of a Data Dissemination Protocol for Network Programming At Scale. In: Proceedings of the Conference on Embedded Networked Sensor Systems, Baltimore, Maryland, November 2004, pp. 81–94 (2004)
11. Krunic, V., Trumpler, E., Han, R.: NodeMD: Diagnosing Node-Level Faults in Remote Wireless Sensor Systems. In: Proceedings of the International Conference on Mobile Systems, Applications, and Services, San Juan, Puerto Rico, June 2007, pp. 43–56 (2007)
12. Levis, P., Brewer, E., Culler, D., Gay, D., Madden, S., Patel, N., Polastre, J., Shenker, S., Szewczyk, R., Woo, A.: The Emergence of a Networking Primitive in Wireless Sensor Networks. Communications of the ACM 51(7) (July 2008)
13. Levis, P., Patel, N., Culler, D.E., Shenker, S.: Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: Proceedings of the Symposium on Networked System Design and Implementation, San Francisco, California, March 2004, pp. 15–28 (2004)
14. Li, M., Yan, T., Ganesan, D., Lyons, E., Shenoy, P., Venkataramani, A., Zink, M.: Multi-User Data Sharing in Radar Sensor Networks. In: Proceedings of the Conference on Embedded Networked Sensor Systems, Raleigh, November 2007, pp. 247–260 (2007)
15. Lorincz, K., Chen, B.-r., Waterman, J., Werner-Allen, G., Welsh, M.: Resource Aware Programming in the Pixie OS. In: Proceedings of the Conference on Embedded Networked Sensor Systems (2008)

16. McIntire, D., Ho, K., Yip, B., Singh, A., Wu, W., Kaiser, W.J.: Low Power Energy Aware Processing (LEAP) Embedded Networked Sensor System. In: Proceedings of the International Conference on Information Processing in Sensor Networks, Nashville, Tennessee, April 2006, pp. 449–457 (2006)
17. Mulligan, G.: The 6LoWPAN Architecture. In: Proceedings of the Workshop on Embedded Networked Sensor, June 2007, pp. 78–82 (2007)
18. Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D.: Sympathy for the Sensor Network Debugger. In: Proceedings of the Conference on Embedded Networked Sensor Systems, San Diego, California, November 2005, pp. 255–267 (2005)
19. Rowe, A., Rosenberg, C., Nourbakhsh, I.: A Low Cost Embedded Color Vision System. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2002, pp. 208–213 (2002)
20. Tolle, G., Culler, D.: Design of an Application-Cooperative Management System for Wireless Sensor Networks. In: Proceedings of the European Workshop on Wireless Sensor Networks, January 2005, pp. 121–132 (2005)
21. Wachs, M., Choi, J.I., Lee, J.W., Srinivasan, K., Chen, Z., Jain, M., Levis, P.: Visibility: A New Metric for Protocol Design. In: Proceedings of the Conference on Embedded Networked Sensor Systems Raleigh, November 2007, pp. 73–86 (2007)
22. Werner-Allen, G., Swieskowski, P., Welsh, M.: MoteLab: A Wireless Sensor Network Testbed. In: roceedings of the International Conference on Information Processing In Sensor Networks, Los Angeles, California, April 2005, pp. 483–488 (2005)
23. 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: Proceedings of the International Conference on Information Processing in Sensor Networks, Nashville, Tennessee, April 2006, pp. 416–423 (2006)
24. Yang, J., Soffa, M.L., Selavo, L., Whitehouse, K.: Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sensor Networks. In: Proceedings of the Conference on Embedded Networked Sensor Systems, Raleigh, November 2007, pp. 189–203 (2007)

Author Index