Sensor Networks: Distributed Algorithms Reloaded – or Revolutions?

Roger Wattenhofer

Computer Engineering and Networks Laboratory
ETH Zurich, 8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

Abstract. This paper wants to motivate the distributed algorithms community to study sensor networks. We discuss why sensor networks are distributed algorithms, and why they are not.

1 Introduction

Wireless sensor networks currently exhibit an incredible research momentum. Computer scientists and engineers from all flavors are embracing the area. Sensor networks are adopted by researchers from hardware technology to operating systems, from antenna design to middleware, from graph theory to computational geometry. Information and communication theorists study fundamental scaling laws such as the capacity of a sensor network. Networking researchers propose new protocols for all layers of the stack. And for the database community, a sensor network essentially is – a database.

The distributed algorithms community should join this big interdisciplinary party! Distributed algorithms are central since – in a first approximation – a sensor network can be modeled as a message passing graph. Hence there is hope that distributed algorithms can be either directly used for or at least adapted to sensor networks.

In the last twenty years, distributed network algorithms have been a thriving theoretical research subject. So far however with limited influence on practice. Sensor networks may be a foremost application area of this vivid theory. Unlike other natural application areas such as the Internet or peer-to-peer/overlay networks, sensor networks are less prone to side effects such as selfish behavior of individual nodes, as generally the whole network is owned by a single entity.[1]

So, can we directly apply our distributed algorithms instruments when developing algorithms for sensor networks? In other words, are sensor networks nothing but distributed algorithms *reloaded*?! In this paper we study to what extent the wireless nature of sensor networks is changing the game. We identify and briefly discuss two modeling aspects for which we believe that sensor networks are fundamentally different from orthodox distributed algorithms.

[1] Interestingly, the other camp of the distributed computing community which deals less with loosely-coupled networks and more with tightly-coupled multiprocessors (a.k.a. shared memory systems) is currently experiencing a similar impetus from the application domain with forthcoming multicore architectures.

P. Flocchini and L. Gąsieniec (Eds.): SIROCCO 2006, LNCS 4056, pp. 24–28, 2006.

First, we need a model which reflects a typical topology of a sensor network. Traditionally, a sensor network is modeled as a *graph*, representing nodes by vertices and wireless links by edges. Geometry comes into play as the distribution of nodes in space, and the propagation range of wireless links, usually adhere to geometric constraints. Several models inspired by both graph theory and geometry are possible; what model is right depends on the question analyzed. A media access study might need a detailed model capturing several low-level aspects. For instance, it has to be taken into account that a message may not be received correctly due to a near-by concurrent transmission. Hence, it is crucial that the model appropriately incorporates interference aspects. However, for a transport layer study, a much simpler model which assumes random transmission errors might be sufficient. In a recent survey [12], a whole zoo of models borrowing from both graph theory and geometry is presented, comprising classic models such as the unit disk graph or the signal-to-interference-plus-noise ratio, but also novel generalizations such as the bounded independence graph or the unit ball graph. These geometric graph models will probably influence the research on distributed network algorithms. For details, we refer the reader to [12].

Second, the very definition of a distributed algorithm is about to change when entering the sensor network domain. We believe that new algorithm types will emerge, and will influence the distributed algorithms community in the coming years. In the remainder of the paper, we briefly discuss possible directions of research.

2 Algorithms

The distributed algorithms community has never been shy of models. We study message passing and shared memory systems, synchronous and asynchronous algorithms, Byzantine and selfish nodes, self-stabilization and failure-detection, to only name a few of the most typical modeling facets. In fact, what is (im)possible and/or (in)efficient *in which model* of distributed computation often outranks the importance of solving this or that problem in a specific model. Still, when it comes to sensor networks it seems that our abundance of models is not enough.

Most algorithms for sensor networks proposed in literature are meant to be *executed by the sensor nodes* during the system's operation. For example, when a node receives a message, it performs some (simple and local) computation, and—depending on the computation's results—sends a new message to its neighbors. A node a priori only knows its own state. In order to learn more about the other nodes in the network, it has to communicate with its neighbors. By collaboration of the nodes, global operations such as (multi-hop) routing can be achieved. Since the activity is distributed among the nodes, these algorithms are called *distributed algorithms* [10]. Distributed algorithms raise many interesting research questions. For example: What can be computed in a distributed fashion, and what not? How efficient is a distributed algorithm compared to a corresponding global algorithm?

Every (global) algorithm can easily be turned into a distributed algorithm: Simply centrally collect the distributed state, compute a global solution, and distribute this solution. However, this simple routine is often unreasonably pricey. Since sending and receiving messages are expensive operations in wireless networks (e.g., medium

access control, energy consumption), a reasonable distributed algorithm should minimize communication. This motivates the introduction of *localized algorithms* [5, 13].[2] A localized algorithm is a special case of a distributed algorithm.

Model 1 (Localized Algorithms). *In a* k-localized algorithm*, for some parameter* k*, each node is allowed to communicate at most* k *times with its neighbors. A node can decide to retard its right to communicate; for example, a node can wait to send messages until all its neighbors having larger identifiers have reached a certain state of their execution.*

In spite of the restricted communication model, localized algorithms can be slow. A node u might have to wait for a neighbor v, while node v in turn has to wait for its neighbor w, etc. Thus, as a matter of fact there can be a *linear chain of causality*, with only one node being active at any time. This yields a worst-case execution time of $\Theta(n)$, where n is the number of nodes.[3] If we do not want this linear running time, we need to resort to another model [8, 10].

Model 2 (Local Algorithms). *In a* k-local algorithm*, for some parameter* k*, each node can communicate at most* k *times with its neighbors. In contrast to* k*-localized algorithms nodes cannot delay their decisions. In particular, all nodes process* k *synchronized phases, and a node's operations in phase* i *may only depend on the information received during phases* 1 *to* $i-1$*. The most efficient local algorithms are often randomized [7, 9]; that is, the number of rounds* k *can vary.*

Observe that in a k-local algorithm, nodes can only gather information about nodes in their k-neighborhood. In some local algorithms [7] the algorithm designer can choose an arbitrarily small constant k (at the cost of a lesser approximation ratio). This makes local algorithms particularly suited in scenarios where the nodes' environment changes frequently, as the algorithm can constantly adapt to the new circumstances. However, due to the synchronous phases, local algorithms may make greater demands on the media access sub-layer than localized algorithms. In particular, in unreliable wireless networks it seems to be costly to implement a media access control scheme that allows for synchronous rounds, as messages will be lost due to interference (conflicting concurrent transmissions) or mobility (even if the nodes themselves are not mobile, the environment is typically dynamic, temporarily enabling/disabling links).

Dealing with unreliability has always been a core interest of the distributed computing community. A powerful concept for coping with failures is *self-stabilization* [4]. Fortunately, using a simple trick [3], every local algorithm is immediately self-stabilizing. The trick works as follows (Section 4 of [3]): Every node keeps a log of every state transition it has taken until its current state; generally this boils down to memorizing the local variables of each step of the main loop. If each node constantly sends its current log to all neighbor nodes, each node can check and correct every transition it has made in the past. Assuming that all inputs are correct (variable initialization and random seeds are stored in the imperishable program memory, sensor information

[2] To the best of our knowledge nobody has ever bothered to formally define what a localized algorithm is. However, all papers we are aware of implicitly use a model similar to Model 1.

[3] And many localized algorithm do exhibit this linear worst-case.

can be re-checked) every fault due to memory or message corruption will be detected and corrected. For details we refer to [3].

Turning a k-local algorithm into a self-stabilizing algorithm with [3] blows up messages by a factor k (in the worst case); on the other hand we immediately get an algorithm which works on a sensor network as the hardest wireless problems (messages lost due to interference and mobility) are covered by the self-stabilization model. Also, in case of an error (such as a lost message), only the k-neighborhood of a node is affected.[4]

In practice, for some local algorithms the detour to self-stabilization may be costly, as the message overhead is prohibitive;[5] instead we need models that integrate interference. One solution is the so-called unstructured radio network model [1, 2, 6] where the algorithm designer has to implement her own medium access scheme from scratch.

Model 3 (Unstructured Radio Networks). *In the* unstructured radio network *model time is divided into slots. In each time slot, each node can decide whether to transmit, listen (or sleep). If two conflicting nodes transmit simultaneously, a potential receiver cannot decode any message. Nodes are distributed in an arbitrary (worst-case) multihop fashion, and may wake-up asynchronously (also worst-case).*

The unstructured radio network model may be classified further, for example depending to what extent collisions can be detected by a receiver. The unstructured radio network model seems to fit practice well, especially if teamed up with a sensible topology/interference model such as signal-to-interference-plus-noise ratio or bounded independence graph [12]. Clearly, the slotted-time assumption is a simplification, however as usual the difference between slotted and unslotted can easily be bounded [11].

Unfortunately, unstructured radio network algorithms tend to be quite technical, as even higher-layer algorithms need to specify media access. We believe that there is room for novel models with more coarse-grained assumptions how the media is accessed. One might for example imagine a model abstracting away from media access, where an adversary schedules transmissions. It seems that this model only makes sense if the adversary is restricted appropriately, that is, if there are fairness guarantees. For example, the adversary might have to schedule each node at least once every $\Theta(n)$ rounds. Moreover, one could imagine an adversary which delivers a message only to a subset of a node's neighbors, because the other neighbors experience collisions.

3 Conclusions

This paper has presented and compared a subjective selection of algorithmic models. For other modeling aspects, we refer to [12]. We want to emphasize that there is no optimal model, and that an engineer has to choose the model which reflects her needs best. Generally, we believe that for efficiency considerations, a slightly idealistic model can be fine. However, when it comes to issues such as correctness of an algorithm,

[4] In principle localized algorithms can also benefit from [3], however, errors are not restricted to a k-neighborhood but may propagate the whole network – we experience a troublesome *butterfly effect*.

[5] Currently the payload constant of a packet in TinyOS is 29 bytes.

it seems that a more pessimistic or conservative model should be preferred. In other words, a *robust* algorithm is also correct in a more general model than for which it has been studied or proven efficient.

Acknowledgments

We would like to thank Stefan Schmid and Thomas Moscibroda for valuable discussions.

References

1. N. Abramson. The ALOHA System. In *Computer-Communication Networks, Prentice Hall*, 1973.
2. N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A Lower Bound for Radio Broadcast. In *Journal of Computer and System Sciences*, 1991.
3. B. Awerbuch and G. Varghese. Distributed Program Checking: A Paradigm for Building Self-stabilizing Distributed Protocols. In *32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991.
4. E. W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. In *Communications of the ACM*, 1974.
5. D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Fifth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1999.
6. F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing Newly Depoloyed Ad-hoc and Sensor Networks. In *10th Annual Intl. Conf. on Mobile Computing and Networking (MobiCom)*, 2004.
7. F. Kuhn, T. Moscibroda, and R. Wattenhofer. The Price of Being Near-Sighted. In *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2006.
8. N. Linial. Distributive Graph Algorithms – Global Solutions from Local Data. In *28th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1987.
9. M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *SIAM Journal on Computing*, 1986.
10. D. Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
11. L. G. Roberts. Aloha packet system with and without slots and capture. In *Computer Communication Review*, 1975.
12. S. Schmid and R. Wattenhofer. Algorithmic Models for Sensor Networks. In *14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Island of Rhodes, Greece*, April 2006.
13. Y. Wang, X.-Y. Li, P.-J. Wan, and O. Frieder. Sparse Power Efficient Topology for Wireless Networks. *Journal of Parallel and Distributed Computing*, 2002.