Energy-efficient Information Collection and Dissemination in Wireless Sensor Networks

by

Zhigang Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2009

Doctoral Committee:

Professor Kang G. Shin, Chair
Associate Professor Jason Flinn
Associate Professor Mingyan Liu
Associate Professor Yaoyun Shi

To my parents, my wife, my daughter, and my sister.

ACKNOWLEDGEMENTS

I owe thanks to many people during this long and bumpy journey in creating this dissertation. First, I want to thank Professor Kang Shin for his guidance and support during my Ph.D.. I am deeply grateful for his willingness to let me pursue almost any research ideas and his great patience for me to seek solutions. His technical insights and words of encouragement will never be forgotten. I would also thank Professors Jason Flinn, Mingyan Liu, and Yaoyun Shi for serving on my thesis committee. Their advice and support helped me improve my work and finish the thesis.

I am most indebted to my family who have all be encouraging and supportive all these years. My parents stood by me from the beginning and believe in me all times. My wife Huaying has unwaivering faith in me. Her love, care and patience, have been tremendous support for me, particularly during some difficult times. My sister Zhimin took care of my parents while I was far away from home, so I could focus on my work with little distraction. My parents-in-law gave me a great deal of help since the birth of my baby so that I can finish the dissertation in time. The thesis is dedicated to them.

I also thank Xin Hu and Min-gyu Cho for collaboration and funny jokes in the days and nights we worked in office. I also thank Jian Wu, Wei Sun, Xin Zhao, Kyu-Han Kim, Alex Min, Hyoil Kim, as well as other RTCL members for their fellowship with me and invaluable comments on my research.

Lastly, I want to thank Julia and John for friendship with my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Wireless Sensor Networks (also known as sensornets or WSNs) are a useful tool to sense and analyze both the physical world and human spaces. Sensors each of which combine sensing, computation, and communication into a resource-constrained device can form an ad hoc network to collect and disseminate information, providing versatile scopes into both physical environments and human activities. While the capabilities of an individual node are limited, the cooperation among hundreds of nodes enables information collection and dissemination for a wide array of applications, such as wild habitat monitoring [1], battlefield surveillance [2, 3], asset tracking [4], road condition and traffic monitoring [5, 6], and people's presence sensing [7, 8].

A representative application of WSNs is environment monitoring. Sensors periodically sense the environment and then aggregate the samples along a spanning tree rooted at a gateway using in-network processing [9, 10, 11, 12]. For example, more than a hundred of MICA mote sensors formed a mesh network to monitor the temperature and humidity of a wildlife habitat [1].

With recent miniaturization and subsequent introduction of sensors into popular consumer electronics, such as cell phones (like Google's G1 and Apple's iPhone) and PDAs (like Apple's iPod Touch), wireless sensor networks have been developed to sense, analyze, and share information about humans, the community, and the way humans live and interact with each other. For example, smart phones are exploited to add sensing presence to people's social network experiences [13], and to monitor road and traffic conditions [5].

The sensor hardware platforms also differ significantly in processing power, radio capability and storage space. For example, a Mica mote has an 8-bit, 4MHz processor, 20Kbps radio bandwidth, and 512 KB storage space, while a G1 has an 32-bit, 528MHz processor, 54Mbps

bandwidth, and a few Gigabytes of storage. However, these battery-powered devices have limited power supply. The lifetime of a Mica sensor is bounded by the capacity of two AA batteries, and cell phones are usually recharged daily or every other day. Although the sensors have been designed with low-power components, it is very challenging to achieve high application performance while minimizing energy consumption.

To meet this challenge, this dissertation has developed energy-efficient information collection and dissemination protocols and systems for WSNs. Specifically, it has explored energy-efficient solutions to the following four problems.

1. Data Aggregation: is crucial for environment monitoring and surveillance applications. Previous studies [9] show that *in-network* data aggregation — intermediate nodes compute partial aggregation results and propagate them towards the base station (BS) or the data sink — is significantly more efficient in terms of communication cost and energy consumption than collecting all sensor readings to the BS which then processes them.

   Aggregation accuracy is affected by the fidelity of computation and communication within the network, but the existing approaches fail to achieve a good combination of both aspects, increasing inaccuracy. TAG [9] computes intermediate aggregation results accurately, but suffers from message losses. Stretch [10] and SD [11] are very robust to message losses, but incur non-trivial computation error.

   In Chapter 2, we explore the design space of data aggregation and develop a novel method for opportunistic data aggregation (OPAG) with zero computation error and good tolerance to moderate message losses, as most of real-world networks have been designed, deployed, and maintained to operate under reasonably good conditions.

   OPAG opportunistically uses multi-path routing to compensate for communication losses and achieve better energy-efficiency than the other schemes using retransmission. This is attributed to a key observation that, when sending a message, the radio (e.g., CC2420, a widely-used, low-power, and high-speed radio) may consume much more energy for idle listening during the backoff period and the time of awaiting the acknowledgment than transmitting the bits. Then, retransmitting a message is not energy-efficient because it incurs more idle listening on backoff and more time to wait for the acknowledgment.

2. Post-Deployment Performance Debugging: besides collecting information about the external world, a sensor network may have to diagnose its own "health" after deployment by disseminating debugging commands and collecting debugging information to locate performance problems. Post-deployment performance debugging is often more difficult and more expensive than pre-deployment debugging in a laboratory setting. When a WSN application exhibits poor performance, users (i.e., programmers and administrators) usually do not know which nodes in the network, nor which pieces of debugging information, should be examined closely to determine the causes of the problems.

   Having every node record and report detailed debugging information (as in Sympathy [14] and MintRoute [15]) may consume a lot of battery power and shorten the network lifetime. It may also cause distraction from the effort of locating the problem sources, and interfere with execution of the underlying applications. NMS (Network Management System) [16] requires users to interrogate individual nodes. But, without any clue regarding the causes of the problems, they may have to query a large number of nodes before locating and isolating the problems, thus taking a long time and consuming a significant amount of energy.

   Therefore, in Chapter 3, we present a data-centric approach called *post-deployment performance debugging* (PDPD or PD2 for short), which helps users locate where performance problems occur, and provides hints for fixing or further investigating the problems. PD2 focuses on the data flows that a WSN application generates, and relates "poor performance" of the application to significant data losses or latencies of some data flows (i.e., problematic data flows) as they go through the software modules on individual nodes and through the network. Based on the data dependencies between different software modules and between different nodes, PD2 derives a few inference rules, and uses them to trace back each problematic flow. PD2 turns on performance monitoring of, and collects debugging information from, only those modules and nodes that the problematic flows go through. Lastly, PD2 visualizes the debugging information to locate the dominant sources (i.e., some software components on certain nodes) of significant data losses and latencies on the data flows.

3. Location Service Protocol: some applications, such as battlefield surveillance, may require a sensor network to detect and report events of interest, like presence of poisonous chemicals,

to mobile data sinks, such as soldiers or vehicles that move within the deployment area. Maintaining a high packet-delivery ratio becomes more difficult as the network gets bigger and the mobile moves faster.

Using location service protocols, a mobile periodically reports its location to selected nodes, called *location servers*. Other nodes can then acquire the mobile's location from one of its location servers and deliver data to the mobile sink using location-based routing methods [17, 18, 19, 20]. There are hierarchical location service protocols (such as GLS [21] and DLM [22]) which construct a hierarchy of location servers over a grid structure, and centralized protocols (such as GHLS [23]) which select a single location server. As the network size increases, the former scales better than the latter, but incurs much higher energy consumption.

In Chapter 4, we develop a distributed location service protocol (DLSP). DLSP can sustain a high packet-delivery ratio in large-scale networks with higher mobile speeds, and incur lower overhead (i.e., lower energy consumption) than other approaches. Through a rigorous analysis, we derive the condition under which a high data-delivery success rate is achieved, and show how to configure the protocol parameters to ensure the scalability of location service. Furthermore, we find that, in order to guarantee a high data-delivery ratio, the mobile's speed should be below a certain fraction of the packet-transmission speed. For example, if the movement threshold for the lowest-level location servers is the same as the node's radio range, the mobile's speed limit is a one-tenth of the packet-transmission speed. The theoretical speed limit is a one-fifth of the packet-transmission speed beyond which DLSP does not scale regardless of the movement threshold. These analysis results can be used for configuring the low-power duty cycles of the sensor nodes when the limit of mobile speeds is known.

Like GLS, DLSP incurs a high location-update overhead. To lower the overhead, we propose an optimization, called *DLSP with a Selected Neighbor* (DLSP-SN), in which the mobile updates the location server in at most one neighbor square at each level. The neighbor square is selected based on the mobile's trajectory. As a result, DLSP-SN achieves both high success rates and low overhead.

4. Location Privacy: by exploiting personal mobile devices as sensors, WSNs provide an extensive view of human spaces. Such a sensing model, called *people-centric sensing*, is used for people and communities' benefits.

   While people-centric sensing can significantly enhance our understanding about human activities and improve user experiences, it raises the concern about users' privacy. Each mobile device can potentially capture a lot of user information, such as activities, habits, and surroundings, from the camera, microphone, GPS, accelerometer, and information on the phone, such as the calendar, music playlists. In particular, a person's location traces may be associated with sensitive places s/he has visited, from which, for example, medical conditions, political affiliations, religious preferences could be inferred.

   To protect users' location privacy, two types of approaches have been proposed. The first type introduces the uncertainty of user identity by masking location information [24, 25]. Specifically, a user's location, cloaked by a geographic area and a time window, is *k-anonymous* if and only if this user and at least $k-1$ other users were present in the area during that time window. So, attackers cannot distinguish among k or more users. The second type introduces "unlinkability" between different pseudonyms of a user via *mix zones* [26]. Users change to new, unused pseudonyms whenever they enter a mix zone, thus mixing their identities. This way attackers cannot link people going into the mix zone with those coming out of it. Both methods require a trustworthy third-party server to count the number of users inside a cloaked or mix zone and ensure that the uncertainty of user identities or the degree of unlinkability between pseudonyms is met.

   In Chapter 5, we explore the tradeoff between privacy preservation and energy consumption, and propose a new approach for protecting location privacy in the context of location-based services. Our approach, called *Location Information ScrAmbler* (LISA), protects location privacy of mobile users using *m-unobservability*. It prevents the distinguishability of a mobile's points of interest (POIs), thereby weakening attackers' ability to infer the user's private information or mobility patterns. LISA adjusts the location noise level in location service queries and ensures that the uncertainty of the attackers' location estimation satisfies *m-unobservability*. LISA protects location privacy locally on each mobile user's handheld

devices and therefore eliminates the need for trustworthy third-party servers. Lowering the trust requirement reduces the risk of leaking privacy information and facilitates deployment of LBSs. However, this also incurs extra energy consumption on mobile devices, so LISA explores the trade-off between the estimation uncertainty and the energy consumption to achieve both strong privacy preservation and efficient energy conservation. Our extensive evaluation using real-world traces of human and vehicle mobility patterns demonstrates the efficacy and efficiency of LISA.

The main contributions of this dissertation are summarized as follows.

- We develop a novel method for opportunistic data aggregation in WSNs, which achieves both high aggregate accuracy and good energy efficiency.
- We propose a data-centric approach for post-deployment performance debugging in WSNs, which helps users locate performance problems effectively and consumes energy efficiently.
- We design a distributed location service protocol (DLSP), which achieves a high packet-delivery ratio in larger networks and at higher mobile speeds, and incurs lower overhead (i.e., lower energy consumption) than other location service protocols.
- We propose a location privacy protection approach called *Location Information Scrambler* (LISA), which protects the location privacy of mobile users effectively and uses battery power of their handheld devices efficiently.

The dissertation is organized as follows. First, Chapter 2 presents the opportunistic data aggregation (OPAG) scheme. Then, Chapter 3 describes a post-deployment performance debugging tool (PD2). Next, Chapter 4 details a distributed location service protocol (DLSP). Finally, Chapter 5 discusses Location Information ScrAmbler (LISA). In Chapter 6 we summarize our main contributions and discuss future directions.

CHAPTER 2

Opportunistic Data Aggregation

2.1 Introduction

In a large-scale wireless sensor network (WSN), data aggregation (such as averaging temperature readings across a network's coverage area) is crucial for such applications as environment monitoring and surveillance. Previous studies (e.g., [9]) show that *in-network* data aggregation — intermediate nodes compute partial aggregation results and propagate them towards the base station (BS) or the data sink — is significantly more efficient in terms of communication cost and energy consumption than routing all sensor readings to the BS which then processes them.

Data-aggregation accuracy in a WSN is affected by the fidelity of both computation and communication within the network. First, computation of aggregation results may be exact or approximate, incurring zero or non-trivial computation error. Second, intermediate aggregation results may be routed via a spanning tree or multiple paths, resulting in different degrees of tolerance to message loss. Figure 2.1 shows different data-aggregation schemes in the design space.

On one side of design space, TAG [9, 27] and Cougar [28] construct a spanning tree rooted at the BS. Each node in the tree combines its own sensor values with the data received from its children, and send the accurate aggregate result to its parent. This spanning tree scheme is simple and works very well for distributive and algebraic aggregates such as MIN, MAX, COUNT and AVG under ideal network conditions. However, its aggregation accuracy degrades significantly when messages are lost during communication. This is because losing a data message from a node leads to the loss of all sensor readings or partial aggregation results from the sub-tree rooted at the node. In order to reduce data loss, each node may dynamically select appropriate parent nodes to

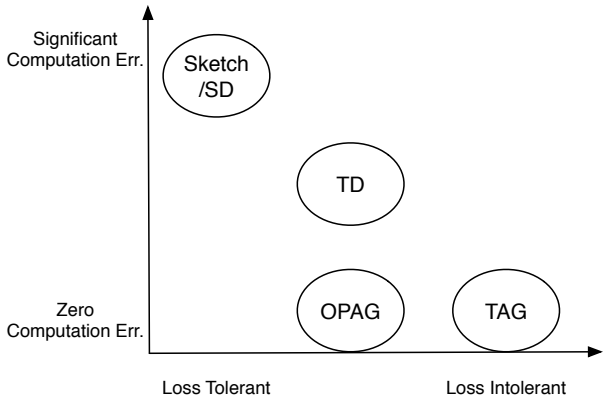


Figure 2.1: Design space of in-network data aggregation according to computation error and tolerance of data message loss. OPAG aims to achieve zero computation error and high tolerance to moderate message losses

avoid using poor-quality links [9], and retransmit lost messages, consuming much more energy.

On the other side of design space, Sketch [10], Synopsis Diffusion (SD) [11], and Tributary-Delta (TD) [12] aggressively exploit multi-path routing to combat message losses. A sensor reading may be aggregated with other readings along multiple paths before it reaches the BS, in contrast to the case of using just one path on the spanning tree. Because of high communication redundancy, Sketch and SD are highly loss-tolerant. However, their multi-path routing is *uncontrolled* — each node has no or little control of which nodes get to aggregate its data, and therefore, a partial aggregation result may be aggregated multiple times into the final result. To deal with duplicate-sensitive aggregates, such as COUNT, SUM, and AVG, statistical counting [29] is used to encode a partial result into a bitmap (a.k.a. Sketch or SD), and convert duplicate-sensitive aggregates to duplicate-insensitive *OR* of such bitmaps. The bitmap obtained at the BS provides an estimate of the aggregation result. The estimation error is non-negligible, and in particular, the variation of estimation is quite high. Compared to other schemes, Sketch and SD work well under a very poor network condition. However, they do not work well for networks under a relatively good condition—which are more common in real-world—because the estimation error persists irrespective of the network condition.

We would like to develop a data-aggregation scheme with both high accuracy and good tolerance to moderate message losses because most of real-world networks have been designed and

deployed to operate under reasonably good conditions. Moreover, the network condition may be improved by adding or upgrading hardware, dynamically switching to channels with less interference, or even altering the deployment. A majority of wireless links are reported to have 0.8 or better delivery probability in both sensor networks [30] and wireless mesh networks [31]. As elaborated on in Section 2.4, our experience with Motelab [32], the WSN testbed at Harvard University, also confirms this observation.

Opportunistic Data Aggregation (OPAG) is a new data-aggregation scheme designed to take advantage of relatively good network connectivity, wherever possible. OPAG allows a node to autonomously choose a *Data-Aggregation Node* (DAN) within a few hops of itself. Then, if the successful delivery ratio over multiple paths to the DAN is above a given threshold, the node has the DAN aggregate its partial results. The DAN aggregates the partial results accurately, so as to avoid computation error, while the other nodes on the multiple paths just relay it. Such *controlled* multi-path routing is likely to be as loss-tolerant as the *uncontrolled* under a relatively good network condition, although it does not provide as much path redundancy. If there is not enough path redundancy to provide a satisfactory success ratio, the node sends the partial results to its parent node as in TAG, and compensates for communication loss by retransmitting them.

OPAG *opportunistically* uses multi-path routing to compensate for communication losses and achieve better energy-efficiency than other schemes using retransmission. This is attributed to a key observation that, when sending a message, the radio (e.g., CC2420, a widely-used, low-power, and high-speed radio) may consume much more energy for idle listening during the backoff period and the time of awaiting the acknowledgment than transmitting the data bits. Then, retransmitting a message is not energy-efficient because it incurs more idle listening on backoff and more time of waiting for the acknowledgment.

In order to avoid extra idle listening, OPAG uses multi-path routing that differs from traditional multi-path routing. Each node dynamically pads multiple partial results and/or sensor readings in a message, i.e., a message may carry multiple partial results each of which traverses a different set of paths. Every receiver disassembles the message and processes the partial results separately— it may aggregate, forward, or discard a partial result, depending on the specified DAN. A node may sometimes have more data to forward than the maximum message payload size, and therefore, has to select a subset of sensor readings or partial results, and drop the rest. A simple selection

strategy may be prioritization of each partial result according to its contribution to the final result —the number of readings over which the partial results are aggregated. However, this may not achieve good aggregation accuracy because high-priority partial results are favored by all forwarding nodes, generating unnecessarily high data redundancy, while low-priority results are unlikely to be forwarded. We analyze the relationship between the data redundancy and the success ratio over multiple disjoint paths, and derive the optimal degree of redundancy, given the accuracy requirement and the number of disjoint paths. Based on this analysis, we design a data-selection algorithm that leads to the optimal redundancy on high-priority partial results and provides as much message space as possible for low-priority results.

This chapter makes the following contributions.

- *Development of a new data-aggregation approach, OPAG*. OPAG achieves improved aggregate accuracy and provides good tolerance to moderate message losses. It opportunistically uses the multi-path routing to compensate for communication losses and achieve better energy-efficiency than those using retransmission.

- *Derivation of optimal data redundancy*. We analyze the relationship between the data redundancy and the successful delivery ratio over a set of disjoint paths, and derive the optimal degree of redundancy with regard to the success ratio threshold and the number of paths. Based on the analysis of this optimal redundancy, we design a data-selection algorithm, which makes a significant improvement of aggregation accuracy.

- *Prototype implementation and experimentation on a real-world testbed and TOSSIM*. We implemented OPAG on TMote Sky motes [33] and conducted experiments on Motelab [32] and simulations over widely-varying network conditions. The results show that OPAG performs much better than TAG and Sketch/SD under relatively good network connectivity.

The rest of this chapter is organized as follows. Section 2.2 illustrates the basic idea of OPAG with an example network. Section 2.3 presents the detailed design of OPAG with emphasis on the energy efficiency. Section 2.4 describes our prototype implementation and presents the experimental results on Motelab, and Section 2.5 presents the simulation results on a large network setup. Section 2.6 discusses the related work. Finally, Section 2.7 concludes the chapter.

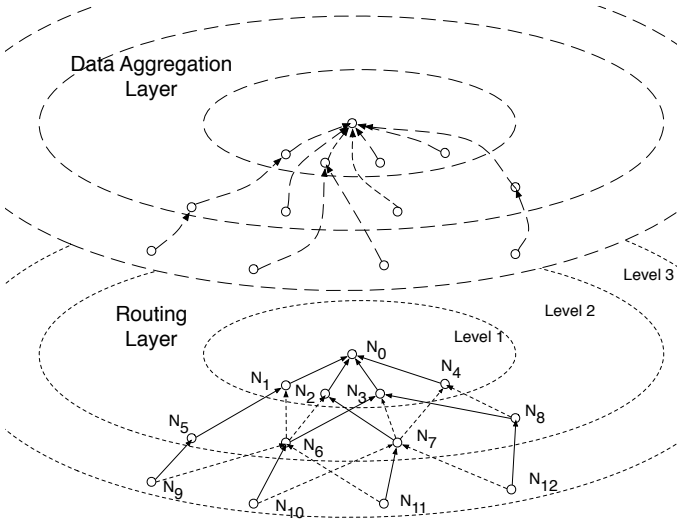


Figure 2.2: Each edge (e.g., $N_{10} \rightarrow N_2$)in the overlay tree shown at the aggregation layer may correspond to multiple paths at the routing layer ($N_{10} \rightarrow N_6 \rightarrow N_2$ and $N_{10} \rightarrow N_7 \rightarrow N_2$).

2.2 Basic Idea

OPAG divides in-network data aggregation into two layers: data aggregation and data routing. At the data-aggregation layer, the aggregation results are computed along an overlay spanning tree. Underneath the routing layer, network nodes may opportunistically send intermediate/partial results via multi-path routing.

Figure 2.2 shows an illustrative example. At the routing layer, the solid edges in the figure indicate child–parent relationships in the spanning tree. Besides the parent node, each node may also communicate with the other neighbor nodes at a lower level in the tree, as indicated by the dotted edges. At the data-aggregation layer, each link in the overlay tree corresponds to a child–parent link or the multiple paths from this node to its DAN.

Neither the spanning tree nor the overlay tree is stored on any node. Instead, it only keeps two data structures that only contain local information. For routing, it keeps a neighbor table that stores a list neighbor nodes and their link quality. The link-quality information is used for computing the delivery probability of forwarding paths. For data aggregation, every node has a list of DAN candidates that are within limited hops and selects the best candidate as its DAN. The details of DAN selection will be presented later.

Suppose N_{10} selects N_2 as its DAN. There are two paths from N_{10} to N_2: $N_{10} \rightarrow N_6 \rightarrow N_2$ and

$N_{10} \rightarrow N_7 \rightarrow N_2$. Therefore, N_2 may receive two copies of N_{10}'s data. The multi-path routing uses data redundancy to combat message losses along either of the two paths. In its partial results, N_{10} specifies N_2 as its DAN. After N_6 receives N_{10}'s message, it extracts the partial results, because the message may contain other partial results forwarded by N_{10} (in this example, we assume N_{10}'s message only contains its own result). Likewise, N_6 receives the data messages from N_9 and N_{11}. N_6 checks each partial result it receives, and aggregates those that specified N_6 as the DAN. Then, N_6 sends the partial results that it should forward as well as its own—N_{10}, N_{11}, and N_6. It discards the partial result of N_9 as N_9's parent node N_5 is to aggregate its partial result. N_9 does not choose N_1 or N_0 to aggregate its partial results because the multiple paths from N_9 to N_1 or N_0 do not provide a satisfactory successful delivery ratio.

After N_2 receives N_6's message, it aggregates the data from N_{10} and N_{11} with its own, because these two data entries specify N_2 as the DAN. Then, N_2's message contains the data of N_6, N_7, as well as the intermediate result of aggregating its own reading and the data from N_{10} and N_{11}. Duplicate data are ignored by N_2.

Because of message space multiplexing, each partial result has more opportunities to reach a specified aggregation node. Thus, OPAG can tolerate message losses without incurring any computation error.

2.3 Design of OPAG

Environment monitoring is usually a long-term application. To achieve a prolonged network lifetime with battery-powered sensors, energy-efficiency is essential to data aggregation, and hence our focus in the design of OPAG.

2.3.1 Energy Cost of Multi-Path Routing and Message Retransmission

Both the multi-path routing and retransmission require the transmission of more bits to tolerate communication losses. So, a natural question is: which one is more energy-efficient?

We use the following example to compare the energy consumption of these two choices. Figure 2.3 shows the energy-consumption breakdowns of multi-path routing and retransmission.

We break down a sender's energy consumption. When an outgoing message is posted on the

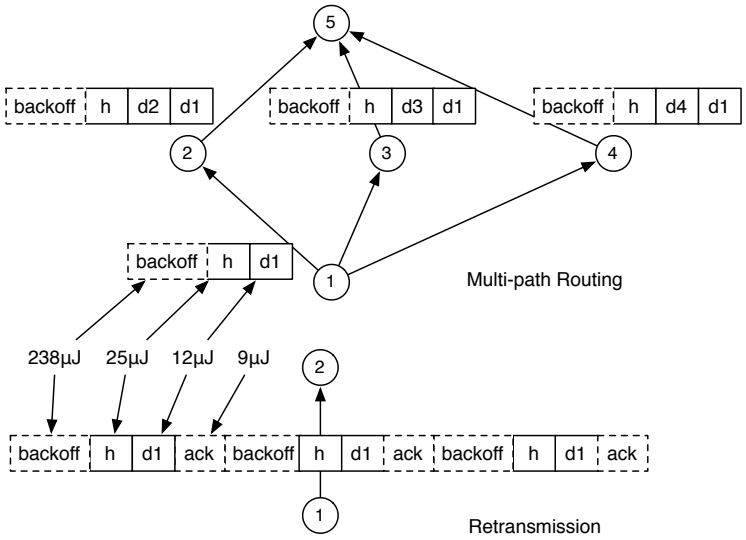


Figure 2.3: The energy costs of the multi-path routing and retransmission using the CC2420 radio stack in TinyOS-2.0.1.

radio stack, the radio is turned on and set to the RX mode for sensing the channel condition (indicated by "backoff"). When the channel becomes clear, the radio starts transmission of the message header (indicated by "h") and the data (indicated by "d1"). After completing the transmission, the sender may turn off the radio, or wait for an acknowledgment (indicated by "ack") if the message is unicast and configured to be acknowledged.

The back-of-the-envelop calculation of the energy costs are based on our experiments using the popular CC2420 radio on the Motelab testbed. We measured the time for the initial backoff, transmitting the payload, and waiting for the acknowledgment. We find that $T_{backoff} = 7.6ms$, $T_{ack} = 2.9ms$, and the time of transmitting d bytes is $d * 0.035ms$. Based on the CC2420 radio datasheet, $P_{rx} = 31.3mW$ and $P_{tx} = 35.5mW$. Therefore, $E_{backoff} = 238\mu J, E_h = 25\mu J, E_{d1} = 12\mu J$ (assuming each partial result has 10 bytes), and $E_{ack} = 9\mu J$.

In the multi-path routing, node 1 broadcasts a message consisting of a common header (h) and its partial result (d1) after a backoff period of waiting for clear channel. Then, the forwarding nodes 2, 3, and 4 broadcast d1 together with their own data (d2, d3, and d4, respectively). Because the forwarding nodes need to send their own data anyway, the backoff and header costs incurred by their data messages are not charged to node 1. Therefore, the total energy cost for d1 is $E_{mp} =$

$E_{backoff} + E_h + 4 * E_{d1} = 311\mu J$.

Using retransmission, if the acknowledgment is not received before the ack timeout, the sender retransmits the message. The sender turns off the radio if the acknowledgment is received, or the retransmission limit is reached. Suppose node 1 does two retransmissions. Then, the total energy cost is $E_{retx} = (E_{backoff} + E_h + E_{d1} + E_{ack}) * 3 = 852\mu J$.

These numbers show that transmitting a few extra bytes consumes much less energy than idle listening during the backoff and the wait for an acknowledgment. Therefore, OPAG chooses the multi-path routing over retransmission, if the multiple paths can achieve a delivery probability above a certain threshold; otherwise, OPAG resorts to retransmission.

The energy consumption on the receiver side depends on receivers' duty-cycling. Previous results [34, 35, 36] have shown that the energy cost of receiving messages depends on a few MAC parameters, such as the preamble length, the polling interval, and the synchronization period. Particularly, with periodic traffic, receiving data may consume about the same amount of energy as sending it. Moreover, the energy consumption of other radio operations, such as turning on/off the radio and switching the radio between RX/TX mode is negligible. Therefore, in this chapter, we focus on the energy consumption of sending data in different data-aggregation schemes.

2.3.2 Selection of Data-Aggregation Node

We now describe how each node selects its DAN. Each node maintains a small list of DAN candidates. A candidate entry contains four attributes: $<id>$, $<level>$, $<p>$, and $<flist>$. The attributes $<id>$ and $<level>$ are the candidate's ID and level in the underneath spanning tree, respectively. $<p>$ is the probability that a candidate node's data is successfully delivered to the data sink through a routing path, and $<p>$ is recursively computed based on the delivery probability of the candidate's DAN, as we will describe later. $<flist>$ is a list of forwarding entries, each of which has two fields: $<fid>$ and $<p_f>$. $<fid>$ is the ID of a neighbor which can forward this node's partial results to the aggregation candidate, and $<p_f>$ is the probability that the neighbor successfully sends partial results to the candidate. Algorithm 1 formally describes the selection process.

We use the example in Figure 2.2 to illustrate how each node maintains its DAN list and selects its DAN. Assume $p(N_1,N_0) = 0.9$, $p(N_2,N_0) = 0.9$, $p(N_3,N_0) = 0.8$, $p(N_6,N_1) = 0.8$,

Algorithm 1 Select a data aggregation node.

Input: A list of DAN candidates (L).

Output: dan, which is the selected DAN.

1: $pmax \Leftarrow 0$
2: $dan \Leftarrow$ null
3: **for all** candidate c in L **do**
4: $p0 \Leftarrow 1$
5: **for all** forwarding node f in c.flist **do**
6: /*get the link quality to the neighbor, fid*/
7: $p1 \Leftarrow$ getNbrOutLQ(f.fid)
8: $pc \Leftarrow 1 - p1 \cdot f.p_f$
9: $p0 \Leftarrow p0 \cdot p1$
10: **end for**
11: $p0 \Leftarrow (1 - p0) \cdot c.p$
12: **if** $p0 \leq pmin$ **then**
13: $dan \Leftarrow c$
14: $pmin \Leftarrow p0$
15: **end if**
16: **end for**

$p(N_6,N_2) = 0.8$, $p(N_6,N_3) = 0.9$. N_6 receives $< 0,0,1.0,0.9 >$, and $< 1,1,0.9,1.0 >$ from N_1; $< 0,0,1.0,0.9 >$, and $< 2,1,0.9,1.0 >$ from N_2; and $< 0,0,1.0,0.8 >$, and $< 3,1,0.8,1.0 >$ from $_3$. Therefore, N_6 has four candidates: $< 0,0,1.0,\{< 1,0.9 >,< 2,0.9 >,< 3,0.8 >\} >$, $< 1,1,0.9,\{< 1,1.0 >\} >$, $< 2,1,0.9,\{< 2,1.0 >\} >$, and $< 3,1,0.8,\{< 3,1.0 >\} >$. The probability that N_6's partial results are successfully delivered via these multiple paths to N_0 is $p_f = 0.98$ (Algorithm 1, line 4–10). So, if N_6 selects N_0 as its DAN, the probability that its partial results are successfully delivered to the sink is $p = p_f * N_0.p = 0.98$ (Algorithm 1, line 11), which is more than using any other candidate. Note that N_6's parent node (can be N_1, N_2, or N_3) is also considered in the selection process.

DAN lists are maintained via DAN announcements, using Algorithm 2 and 3. A node N's DAN announcement contains a few DAN candidates that it wants to advertise. Each entry in the announcement has <*id*>, <*level*>, <p>, and <p_f>. The first three fields are from the corresponding DAN entry, and <p_f> is computed using the forwarding list and the neighbor table (Algorithm 2, line 3–9). A DAN is re-announced only if the change of delivery probability is above a certain threshold (Algorithm 2, line 14–16) and the announcement propagates upward limited hops (Algorithm 2, line 17–19), in order to to control storage and communication overheads. If the DAN candidate is 3 or more more hops away, then the paths from N to the candidate may not be disjointed. Because exact calculation of the aggregate probability of the forwarding paths is complicated and requires propagation of additional topology information, which incurs extra energy cost, we treat them as disjointed and then discount the resulting probability by a factor of 0.8 (line 20-22). Each node also announces itself as a DAN candidate (Algorithm 2, line 29–33). Its own delivery probability is calculated based on the delivery probability of its current DAN (Algorithm 2, line 10–13).

Each node should only consider the DAN announcements from other nodes of a lower tree level, and ignore the announcements from nodes of the same or higher level (Algorithm 3, line 1–3), because only a lower-level node can be its DAN. For example, N_1's announcement should be processed by N_9, but ignored by N_2.

Algorithm 2 Create and send DAN announcements

Input: A list of DAN candidates (L) and the current level of the node, l, my current DAN dan, the limit of hops within which the announcements propagate, h_d, and the threshold of probability difference, p_d.

Output: none

```
1: msg = null
2: for all candidate c in L do
3:     p0 ⇐ 1
4:     for all forwarding node f in c.flist do
5:         p1 ⇐ getNbrOutLQ(f.fid)
6:         pc ⇐ 1 − p1 · f.p_f
7:         p0 ⇐ p0 · p1
8:     end for
9:     p_f ⇐ (1 − p0)
10:    if c is dan then
11:        Me.p ⇐ c.p · p_f
12:        Me.p_f = p_f
13:    end if
14:    if (c.p · p_f − c.prevp) ≤ p_d then
15:        continue
16:    end if
17:    if (c.level − l) ≥ h_d then
18:        continue
19:    end if
20:    if (c.level − l) ≥ 3 then
21:        p_f ⇐ p_f · 0.8
22:    end if
23:    /*this candidate should be re-announced*/
24:    c0 ⇐ (c.id, c.level, c.p, p_f)
25:    insert c0 into the current beacon message or msg has space (create a new message if msg is
       null)
26:    c.prevp ⇐ c.p · p_f
27: end for
28: /* announce this node as a candidate */
29: if (Me.p − Me.prevp) > p_d then
30:    c0 ⇐ (Me.id, Me.level, Me.p, 1)
31:    insert c0 into the current beacon message or msg has space (create a new message if msg is
```

Algorithm 3 Process a DAN announcement

Input: a DAN candidate c and the current level l

Output: none

1: **if** $c.level \geq l$ **then**
2: ignore c
3: **end if**
4: insert c into the candidate list

2.3.3 Selection of Partial Results

The maximum message payload size limits the number of partial results that can be "padded" in a single message. When a node has more results than the message space permits, how does it choose which results to pad in the message? A simple answer to this question may be prioritization of each partial result according to its contribution to the final aggregation results (i.e., the content of the field $<Count>$), thus favoring the partial results with higher priority (i.e., larger counts). However, this may not achieve good aggregation accuracy since some partial results with large counts may be favored by all forwarding nodes, resulting in unnecessarily high degree of redundancy, while those with small counts have little chance to be forwarded. In order to allocate resource (i.e., message space) efficiently, we need to find the relationship between the redundancy and the successful delivery ratio over multiple paths.

Suppose there are k independent (or disjoint) paths from node N and its DAN. Let p be the success probability over the paths (i.e., the probability that N's results are successfully received via at least one of the paths), and d the redundancy (i.e., the average number of duplicates that the DAN receives). The successful delivery probability of the i-th path is denoted as p_i $(i = 1, 2, \cdots, k)$. Then, we have $p = 1 - \prod_{i=1,\cdots,k} (1 - p_i)$ and $d = \sum_{i=1,2,\cdots,k} (p_i)$. The mean and standard deviation of p are, respectively:

$$\begin{aligned} \mu_k(p) &= \sum_{R} (1 - \prod_{i=1,2,\cdots,k} (1 - p_i)) \\ \sigma_k^2(p) &= \sum_{R} (p_k(d) - \mu_k(d))^2 \end{aligned}$$

where $R = \{p_1, \cdots, p_k | d = \sum_{i=1,2,\cdots,k} (p_i) \wedge 0 \leq p_i \leq 1 \, (i = 1, \cdots, k)\}$.

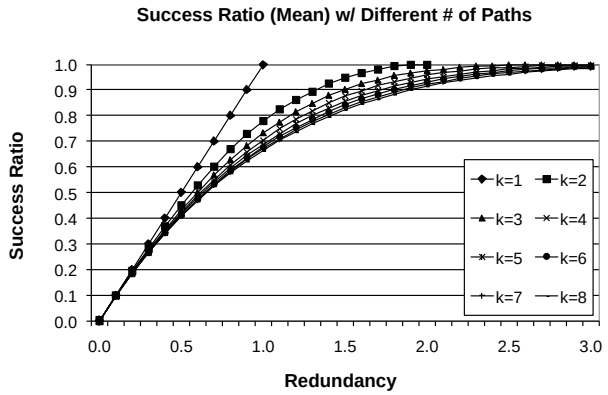


Figure 2.4: μ_p with regard to d and k, where $1 \leq k \leq 8$ and $0 \leq d \leq 3$

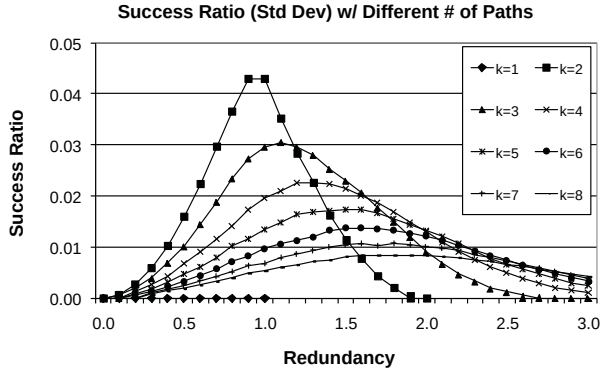


Figure 2.5: σ_p with regard to d and k, where $1 \leq k \leq 8$ and $0 \leq d \leq 3$

Since no closed-form solutions for $\mu_k(p)$ and $\sigma_k(p)$ exist, we find numerical solutions, as shown in Figures 2.4 and 2.5.

Our observation from the analysis is the existence of *appropriate* redundancy that can guarantee a given success ratio, and message space is wasted if this redundancy limit is exceeded. Therefore, we can set the success ratio for a partial result using its contribution to the final result, and then set the forwarding probability, FP, on each forwarding path to enforce the *appropriate* redundancy for that partial result. $FP = \frac{RDD}{d}$, where RDD denotes the *appropriate* redundancy to achieve the given success ratio. If multiple paths favor a certain partial result, its redundancy at the DAN d may exceed the redundancy limit. Then, $FP < 1.0$, which throttles the paths to probabilistically drop the partial result to achieve the optimal redundancy. When $d \leq RDD$, the optimal redundancy has not yet been achieved. Then ,$FP = 1$, letting every path attempt to achieve as high a success ratio as possible.

To enforce the appropriate redundancy, OPAG takes the following steps.

1. After a node computes its partial result, it evaluates the importance of the result based on its tree level and other results it receives, then decides a target success ratio (p) according to the importance—the number of sensor readings the partial result has aggregated.

2. It selects the best DAN from the candidate list and counts the number (i.e., k) of paths via which the DAN can be reached, and estimates the expected redundancy, using the information in the candidate list and the neighbor table (i.e., d).

3. With p and k, the target redundancy RDD can be looked up from a table like Table 2.1—the rows show different success ratios and the columns indicate the number of disjoint paths. Path disjointness holds if the DAN of a node is restricted to be within 2 hops. Otherwise, we get an estimate using neighbor table information and some heuristics.

4. FP is set to RDD/d if $RDD \leq d$, or 1 otherwise.

5. A forwarding node prioritizes the partial results by the COUNT field. Then, it starts from the partial result with the highest COUNT, and selects each partial result according to the probability, FP.

k	1	2	3	4	5	6	7	8
$d_s(p=0.95)$	0.95	1.5	1.8	1.9	2.1	2.2	2.3	2.3
$d_s(p=0.9)$	0.9	1.4	1.5	1.6	1.7	1.8	1.9	2.0
$d_s(p=0.8)$	0.8	1.1	1.2	1.3	1.3	1.4	1.4	1.4

Table 2.1: The appropriate redundancy for three target success ratios with different numbers of paths. The numbers are extracted from Figure 2.4.

Applying the same probability FP on every path is a simple way of enforcing the *appropriate* redundancy. We can improve the efficiency of message space by considering the uneven utilization of each path, but exchanging relevant information for the optimization may incur extra overhead.

2.3.4 Overhead of Maintaining the Overlay Tree

OPAG incurs both communication and storage overheads in maintaining the overlay tree of data aggregation.

The communication overhead represents the DAN announcement messages. This overhead is very small when the network connectivity is relatively stable. This is because the total number of DAN announcements is significantly reduced by using the threshold of the delivery probability and most DAN announcements can be piggybacked in beacon messages. When the network condition becomes more dynamic, more announcements need to be sent to maintain the overlay tree.

The storage overhead incurs for the DAN candidate list and the data buffers. OPAG only keeps a few good DAN entries, so the space cost is small. Each node needs a data buffer to store the partial results it has to forward. Since all DANs are within a few hops, there are only a limited number of partial results a node may need to forward. In very dense networks, a node can randomly pick a few partial results it may consider to forward based on the availability of space.

2.4 System Implementation and Evaluation

This section first details the architecture of our implementation of OPAG, then describes our evaluation methodology and experimental setup, and finally presents our experimentation results.

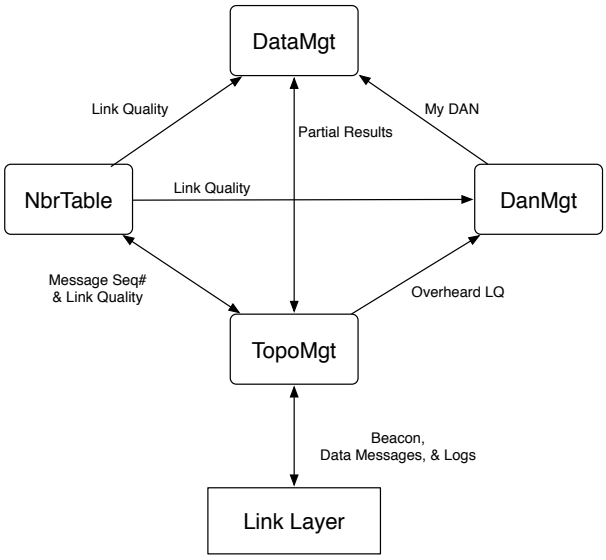


Figure 2.6: Architecture of OPAG implementation on TMote Sky Node.

2.4.1 Implementation Details

We implemented OPAG on the TinyOS 2.0.1, and TMote Sky platform which is equipped with an 8MHz TI MSP430 processor, 10K RAM, and a 250Kbps Chipcon radio operating at 2.4GHz. It supports various sensors, such as light, temperature, and humidity.

Our implementation of OPAG is based on the Collection Tree Protocol (CTP) in TinyOS-2.x source[37]. As shown in Figure 2.6, OPAG consists of four major components: TopoMgt (Topology Management, a modified version CtpRoutingEngine), NbrTable (Neighbor Table, slightly modified from LinkEstimator), DanMgt (Data Aggregation Node Management), and DataMgt (Data Management, modified from CtpFowardingEngine).

Like CtpRoutingEngine, TopoMgt estimates and exchanges link-quality information by periodically broadcasting beacon messages. Then, based on the link-quality information, TopoMgt forms a spanning tree. However, the three changes that TopoMgt makes to CtpRoutingEngine are: (1) each beacon message bears the current clock to achieve a loose time synchronization; (2) the beacon timer is modified so that each node only broadcasts beacons during the beacon slots, and after each beacon message, a DAN announcement message is sent; (3) in addition to using bidirectional link quality, TopoMgt skips those neighbors with poor outgoing link quality when selecting a parent node.

DataMgt extracts the partial results from incoming data messages, and then processes them in three different ways: (1) if a partial result's DAN is the current node, DataMgt aggregates it; (2) if the result's DAN is in the DAN candidate list, DataMgt discards it, because this node would have received some advertisements of the DAN if it is on a forwarding path; (3) for the other partial results, DataMgt selectively forwards some of them, depending on the availability of space in the message payload field.

DanMgt maintains a list of DAN candidates and computes the probabilities to select the best DAN. NbrTable maintains the neighbor table and link-quality information, just like LinkEstimator.

Because multi-path routing is more energy-efficient than retransmission, a node exploits the multi-path routing opportunistically. If the best DAN can meet a given probability requirement, it sends its partial result in a broadcast message, and therefore, the partial result may be forwarded via multiple paths; otherwise, it sends the partial result in a unicast message to its parent node. In the former case, broadcast messages are not acknowledged; in the latter case, the parent node only sends an acknowledgment if it successfully receives the child node's partial result, rather than the entire message.

To support the acknowledgment of a specific partial result rather than the entire message, we added in the CC2420 radio stack a segment CRC checking [38]: instead of having one CRC checksum for the entire message, a 1-byte CRC checksum is added to each of the partial results within the message. The original CRC checksum only verifies the message header. Upon receiving a unicast message from a child node, the parent node first checks the header's CRC. If the header is corrupted, the whole message is dropped. If the child's partial result was received correctly, the parent node sends an acknowledgment. Other partial results, if received correctly, are processed as usual, or discarded if corrupted. The child node needs to retransmit the unicast message unless it receives the acknowledgment of its partial result or the retransmission limit is reached. Other nodes overhear the unicast message and process the partial results individually.

Aggregation query and accuracy: In this chapter, we focus on commonly-used, duplicate-sensitive aggregates, such as COUNT, SUM, and AVG. The size of a partial result for these simple aggregates is usually small, so a data message can hold several partial results, thereby making message multiplexing possible. In our experiments, we run the COUNT aggregate over the entire network, i.e., counting the number of live sensors in the network.

We use the relative error to evaluate the performance of different aggregation approaches. The relative error is defined as $\frac{|x-\hat{x}|}{|x|}$, where $\hat{x}$ is a result and x the true value. In the figures of aggregation accuracy, unless otherwise noted, we show the median values as well as the 5 and 95 percentiles. We do not use the percentage contribution because it does not reflect the computation error incurred by statistical counting in SD.

For complicated queries, such as histogram and Sketch of the light readings of the entire network, the size of a partial result may vary, depending on the amount of data involved and the accuracy of the partial result [39, 40]. So, there is contention for limited message space between the accuracy of a partial result itself, and the need for message multiplexing to tolerate message losses. Extension of OPAG for complicated queries is left as our future work.

Energy-efficiency: Each node stays awake throughout the beacon slot and its receiving slot, and hence the difference is the energy it spends on sending data messages. So, we measured the time of sending data messages and converted it to energy consumption.

We timestamped a few points in the CC2420 radio stack and measured the time each node spends in the TX/RX mode during the sending slots. To obtain the energy consumption, we multiply the time spent in the RX mode by $P_{rx} = 35.5mW$ (i.e., the receive power), and the time in the TX mode by $P_{rx} = 31.3mW$ (i.e., the maximum transmit power). We also used a lower transmit power, $P_{tx} = 22.5mW$ in our evaluation.

2.4.2 Experimental Setup

For the purpose of performance comparison, we implemented OPAG, OPAG-RC (OPAG with the redundancy control, i.e., using the data-selection scheme), TAG (TAG's tree-based approach), TAG-RETX (TAG with retransmission), and SD (Synopsis Diffusion over the ring structure), based on the CTP source. Because the level-1 nodes are just one hop away and cannot establish multiple paths to the BS, SD allows these nodes to send each data message three times [11]. For a fair comparison, the level-1 nodes in TAG also send each data message up to three times. TAG and SD do not retransmit lost messages except for the level-1 nodes. TAG-RETX retransmits a message unless it is acknowledged or there have been 4 retransmissions. We do not include Sketch, because it is almost identical to SD, except that SD uses 32-bit bitmaps, while Sketch uses 16-bit bitmaps.

To compensate for message losses, OPAG and OPAG-RC use multi-path routing if the redundant paths give a success ratio no less than p, and use retransmissions if there is no redundant path or the multiple paths yield a success ratio less than p. A node sets p to 0.95 if it has aggregated the sensor readings of more than 3 nodes, and 0.9 otherwise. Therefore, the partial results with more contribution to the final aggregation result are unlikely to be lost.

All of the approaches under consideration follow the same schedule as follows. Each epoch is 120s long and consists of 24 slots. A common slot is assigned for all nodes to exchange beacon messages. Each node is assigned a receiving slot to receive partial results from other nodes, and a sending slot to send its own partial result (and the partial results it has to forward under OPAG and OPAG-RC), depending on the node's level in the topology.

To evaluate our implementation, we used Motelab, the WSN testbed at Maxwell Dworkin Laboratory of Harvard University. Motelab has about 190 Tmote Sky nodes scattered across a number of rooms on three floors, and about 65 nodes can be programmed. The environment should have enough multi-path effects from obstacles and interference from other wireless communications as a realistic WSN deployment does.

Each node uses a CC2420 radio operating at the default radio frequency—channel 11 of IEEE 802.15.4 (2.405 GHz) with the maximum transmit power. It uses the CC2420 radio stack in TinyOS-2.0.1, which sends 802.15.4-compliant packets, but does not implement the 802.15.4 MAC protocol. The physical header and the MAC header use 20 bytes. The message payload length is limited by the size of the physical data buffer (120 bytes). We set the maximum payload size to 60 bytes. Each partial result is 10 bytes long, including 1-byte filtering probability (FP) and 1-byte CRC checksum.

For the results presented in this chapter, all DAN advertisements are propagated within 2 hops for the following reason. When the DAN is located 3 hops or more away, the multiple paths from a node to that DAN are very likely braided. Instead of deriving the accurate success ratio over the braided paths, we simply multiply 0.8 to the ratio, assuming that the paths are independent of each other. Then, using the estimated success ratio, a node very rarely selects its DAN from 3 hops or more away.

We ran each test 5 times, and each run lasted about 25 minutes. The first 5 minutes is the warmup period, in which nodes build up a spanning tree. Then, the COUNT aggregate query runs

for 8 epochs, each lasting 2 minutes.

2.4.3 Experimental Results

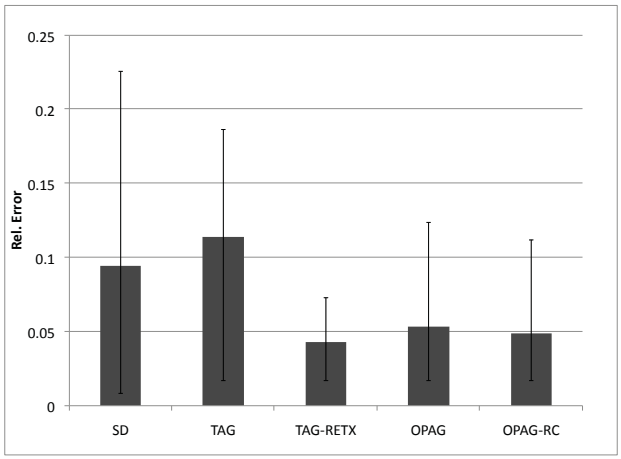
Aggregation Accuracy

We used two transmit powers: 22.5 mW and 31.3 mW (i.e., maximum transmit power), corresponding to -7 dBm and 0 dBm, respectively. With the maximum transmit power, 59 nodes join the spanning tree, while using a lower transmit power, 42–49 nodes join the tree, depending on different runs. Therefore, we have a topology of relatively strong connectivity (Topology-S) with an average node degree of 9.7, and a network diameter of 6, and another of relatively weak connectivity (Topology-W), with an average node degree of 6.3 and a network diameter of 9.2.

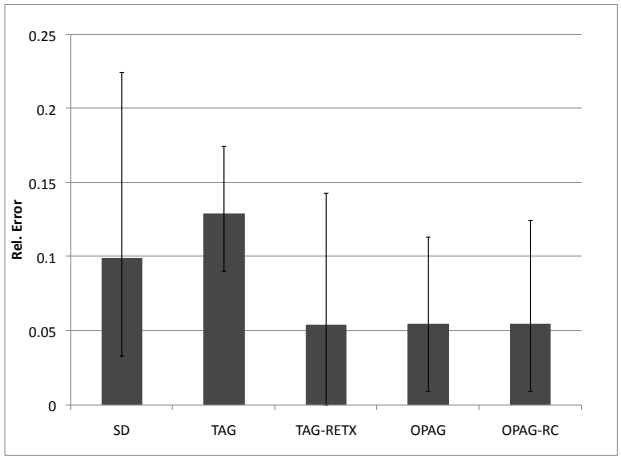
Figures 2.7(a) and 2.7(b) show the aggregation accuracy of the five schemes. In both topologies, TAG has the worst average relative error due to message losses. Using retransmissions, TAG-RETX can reduce the error by 50-65%, as compared to TAG. Compared to TAG-RETX, OPAG and OPAG-RC incur a slightly higher error in Topology-S and a comparable error in Topology-S. This is because in Topology-S, the nodes have better connectivity and thus provide more redundant paths, and OPAG and OPAG-RC use multi-path routing more aggressively. Therefore, with a small probability, a partial result may be lost on all paths, leading to a slightly higher aggregation error, as compared to TAG-RETX. By contrast, in Topology-W, OPAG and OPAG-RC exhibit a relative error similar to that of TAG-RETX, because the nodes are less connected and thus less likely to use multi-path routing.

In both topologies, OPAG and OPAG-RC incur a similar error because there is plenty of message space, i.e., a message can hold as many as 6 partial results. So, the redundancy control makes little difference in aggregation accuracy.

SD is very insensitive to the change of topology and network connectivity because (1) its multi-path routing is extremely aggressive, and (2) the variance of estimation is large. More than 5% of the estimated results differ from the true value by more than 20%.



(a) Topology-S



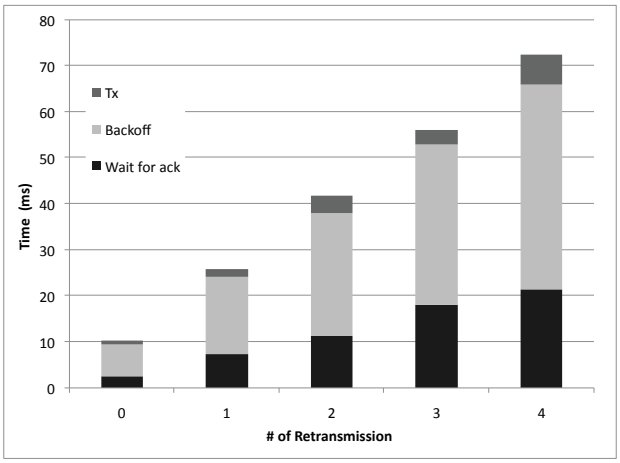
(b) Topology-W

Figure 2.7: Aggregation accuracy

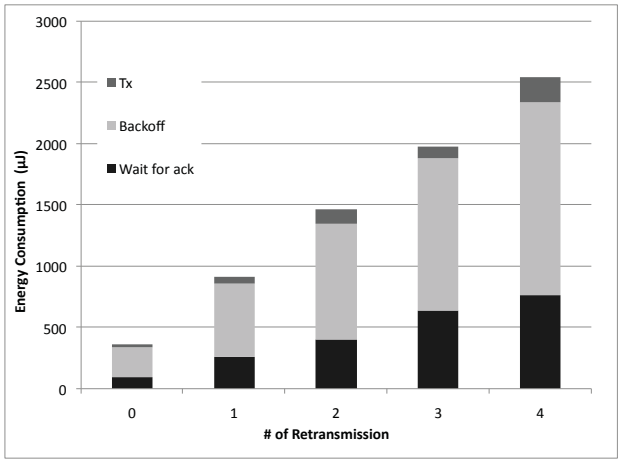
Energy-Efficiency

Before analyzing energy-efficiency, we first show the breakdown of time and energy spent in sending a message in Topology-S by using the number of retransmissions in Figures 2.8(a) and 2.8(b), respectively. It illustrates that the energy cost increases significantly as more retransmissions are attempted. In fact, as shown in Figure 2.8(b), idle listening (in the backoff and the ack waiting) takes more than 90% of the energy cost, and sending a message with 4 retransmissions consumes 7 times more than sending a message without retransmission. Similar results are obtained for Topology-W.

Because the time for transmitting data is much less than the backoff time and the time for wait-



(a) Time Breakdown

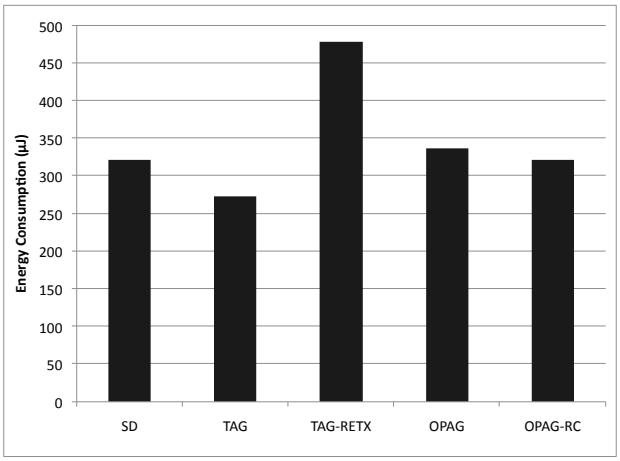


(b) Energy Breakdown

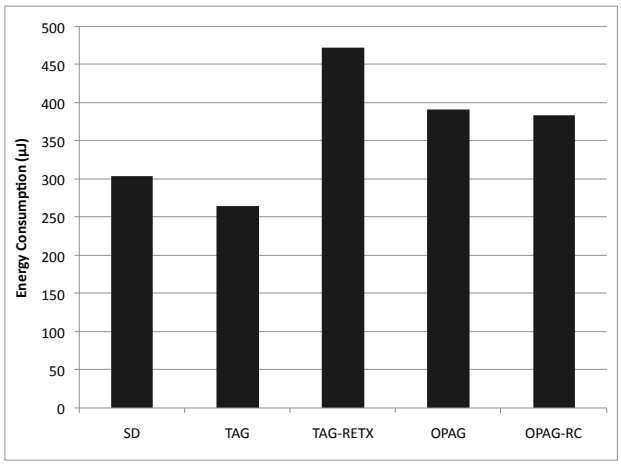
Figure 2.8: The time and energy breakdown with different # of retransmissions in Topology-S

ing for acknowledgments, it is more energy-efficient to let the forwarding nodes on the multiple paths send a few extra bytes than retransmitting the lost messages.

In Figures 2.10(a) and 2.10(b), we show the distribution of transmitted messages with the number of retransmissions. In both topologies, about 88% of messages are sent successfully without retransmission. Due to better connectivity in Topology-S, OPAG and OPAG-RC can take advantage of multi-path routing and increase this percentage to about 95% at the expense of slightly lower accuracy. The energy cost is reduced by about 33% compared to TAG-RETX, and is about the same as SD, as shown in Figure 2.9(a). This indicates that our opportunistic schemes can cut the aggregation error by half at roughly the same energy cost in networks of good connectivity. In



(a) Topology-S



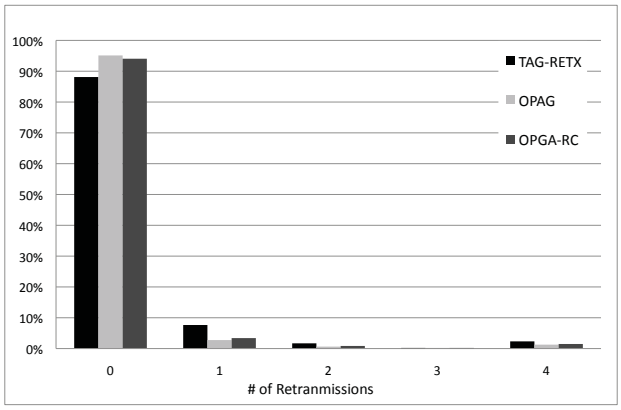
(b) Topology-W

Figure 2.9: The average energy consumption for sending data per epoch per node

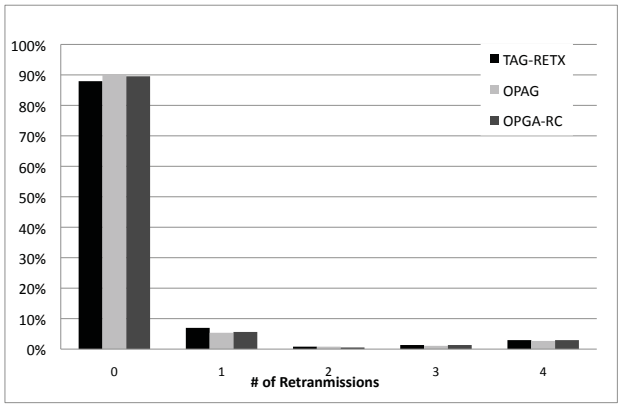
Topology-W, because there are less opportunities to exploit multi-path routing, the energy cost is only down by about 14% as shown in Figure 2.9(b).

In Topology-S, OPAG-RC consumes a little less energy than OPAG because the redundancy control can slightly reduce the amount of data some forwarding nodes have to send. In Topology-W, the difference between OPAG and OPAG-RC is negligible because there is less chance of using multi-path routing and therefore, the redundancy control is less involved.

In both topologies, TAG consumes the least amount of energy because each node sends one short message without retransmission. SD consumes more energy than TAG because it needs to transmit more bytes in each message.



(a) Topology-S



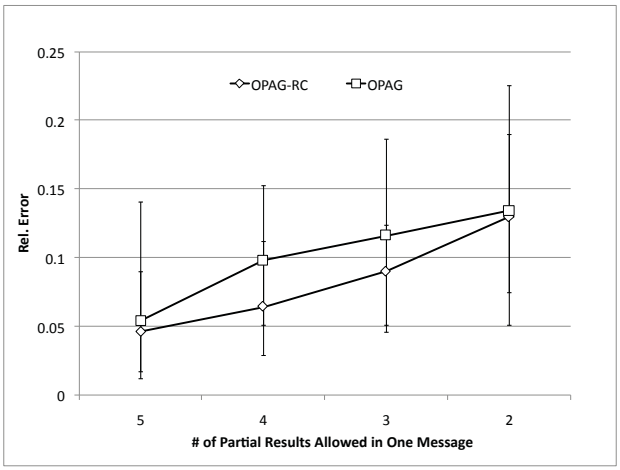
(b) Topology-W

Figure 2.10: The distribution of retransmission

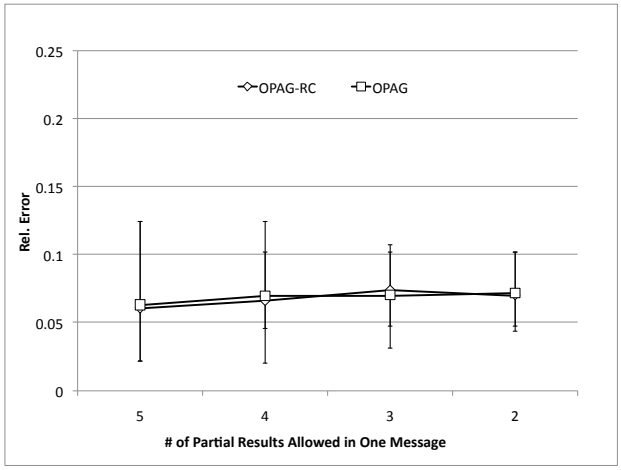
Moreover, Figures 2.8(b) and 2.10(a) show the ability of OPAG and OPAG-RC to balance the energy consumption, because some nodes can avoid retransmissions which incur much higher energy consumption, and the forwarding nodes only need to consume a little more energy for transmitting more bits than retransmitting messages.

Effectiveness of Redundancy Control

To evaluate the effectiveness of redundancy control, we set the maximum number of partial results packed in one message to be 5, 4, 3, 2. Figure 2.11(a) shows the relative errors of OPAG and OPAG-RC. For Topology-S, the redundancy control makes a noticeable difference at the limit of 3 and 4, because several level 2 and 3 nodes need to forward the partial results of about 4 or 5 nodes. In fact, a majority of forwarding nodes only need to send 2 or 3 partial results. At the limit



(a) Topology-S



(b) Topology-W

Figure 2.11: The effectiveness of the redundancy control scheme

of 1 and 5, the message space is either scarce or abundant, the redundancy control is not effective.

In Figure 2.11(b), for Topology-W, multi-path routing is less likely used, so there is little competition for message space, and the redundancy control rarely gets involved.

2.5 Simulation Results

In order to evaluate its performance under widely-varying network conditions, we simulated OPAG using the TinyOS simulator (TOSSIM) [41].

400 nodes are placed evenly in an $80m \times 80m$ grid with node 0 (a.k.a. the BS) at the center, and

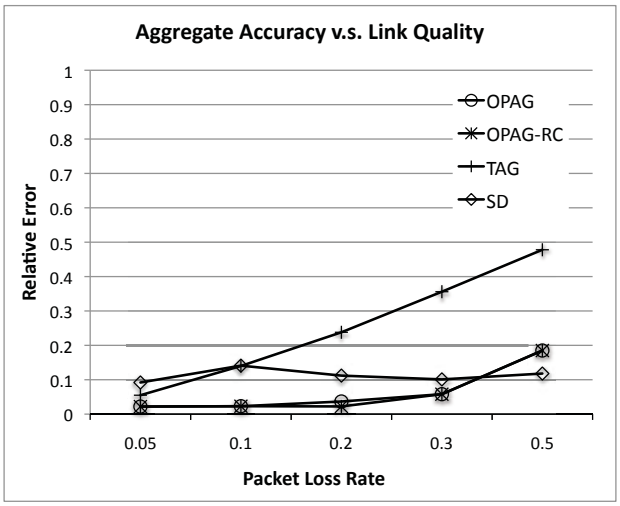


Figure 2.12: Effect of link quality on the aggregate accuracy.

all nodes have a 10m radio range. Then, in level 1 there are three times as many nodes as in level 2. Even if a node at level 2 only uses 2 forwarding nodes in level 1 (i.e., two paths to reach the BS) on average, each node in level 1 has to forward 6 partial results. Therefore, due to limited message space, level-2 nodes are more restricted than those in other rings in exploiting multi-path routing, So, we placed 20 more nodes in the BS's radio range, which basically doubles the node density in level 1. Deploying more nodes around the BS can be justified as they usually drain their batteries faster than those in the rest of the network.

Message size and transmission model: For TAG and SD, the message payload size is 48 bytes, as in previous studies. For OPAG and OPAG-RC, the payload size varies to be 38 bytes, 46 bytes or 54 bytes which hold 4, 5, and 6 data entries, respectively.

We use the bit-level lossy model in TOSSIM, meaning that the size of the message affects the packet loss ratio. Specifically, given the bit-error probability, e_b, the packet loss ratio is modeled as $e_p = 1-(1-e_b)^L$, where L is the total number of bits in a message, including the preamble and the message header.

2.5.1 Effect of communication loss

Figure 2.12 shows the effects of link loss on the aggregate accuracy of different aggregation schemes. We only present the results of OPAG and OPAG-RC with the message payload size of 46 bytes, which is close to the payload size (48 bytes) used in TAG and SD.

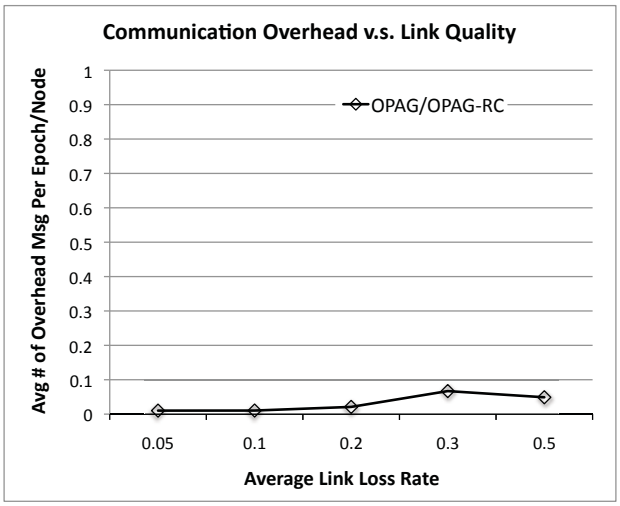


Figure 2.13: Effect of link quality on the overhead of maintaining the overlay tree in OPAG.

We can see that the median relative error for OPAG and OPAG-RC is below or around 0.1 until the link loss rate increases to 0.5. This shows that OPAGs are robust to moderate message losses. The performances of OPAG and OPAG-RC are similar in these tests because the message payload size (46 bytes) accommodates up to 5 data entries, so contention for message space is relatively modest and the data-selection scheme makes little difference.

SD is even more robust to message losses—its median relative error remains around 0.1 even when nearly a half of messages are lost. This is because SD aggressively exploits the multiple paths from each node to the BS, which are much wider (i.e., contains more paths) than the paths from a node to its DAN in OPAG and OPAG-RC. However, due to the estimation error, the relative error does not decrease as the network condition is very good. Because TAG does not deal with messages losses at all, the error increases significantly as the network condition gets worse.

Figure 2.13 shows the effect of network condition on the communication overhead of DAN announcements. The overhead is measured by the average number of DAN advertisement messages per node per epoch, excluding the DAN advertisements during the warmup period. As the average link loss rate increases, the link quality suffers greater fluctuations because the radio model in our simulation randomly "corrupts" packets based on the loss rate. The figure confirms that more fluctuations trigger more updates. As the link quality continues to degrade, the overhead decreases, because many links are labeled as poor links, which are simply ignored in the neighbor tables and therefore, do not trigger any announcements.

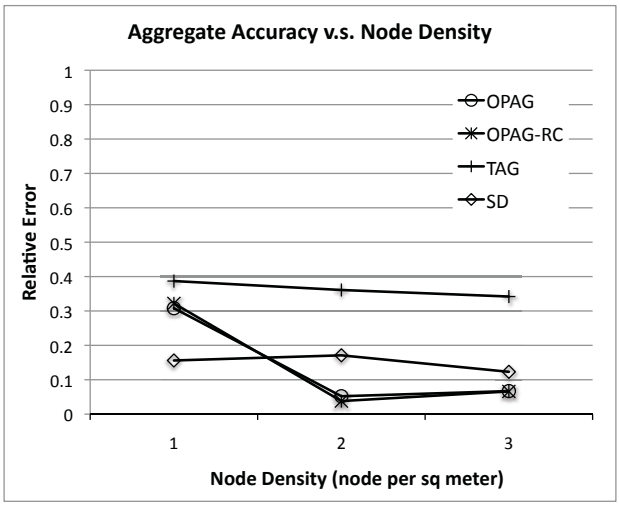


Figure 2.14: Node densities $\frac{1}{64}$, $\frac{1}{32}$, and $\frac{1}{16}$ indicate about 5, 10, and 20 neighbors, respectively.

2.5.2 Effect of node density

To evaluate the impact of node density, we vary the total number of nodes without changing the deployment region. Also, the node density applies to the entire region except for the area within the BS's radio range, in which the density always doubles. Figure 2.14 shows that both OPAG and OPAG-RC are more sensitive to node density than SD for two reasons. First, SD exploits multi-path routing more aggressively than OPAGs. Second, SD's statistic counting is relatively insensitive to data losses due to its estimation error. From Figure 2.14, we find that OPAG performs well when each node has 10 or more neighbors.

2.6 Related Work

A number of researchers studied data aggregation in WSNs [9, 27, 28, 42, 11, 10, 12]. Their approaches use a tree topology (value-splitting can be considered as a special approach based on a tree topology) with exact computation of aggregate results, a ring topology with statistical estimation, or both. The tree-based approaches [9, 27, 28, 42] do not incur any computation error, but are not robust to message losses. In contrast, the ring-based approaches [11, 10] are very robust to message losses by aggressively exploiting multi-path routing, but their statistical estimation leads to significant result inaccuracy.

To combine the advantages of tree topologies and multi-path routing, Tributary-Delta (TD)[12]

allows a WSN to form a hybrid topology, visually termed as *tributary-delta*, where a *tributary* refers to an area where nodes in a very good condition form a sub-tree, and the *delta* refers to the area where the BS and some nodes form a multi-path sub-graph. When the network condition changes, the delta area and the affected tributary areas can be adjusted adaptively subject to the restrictions of topological correctness. TD can significantly improve the accuracy of aggregate results over that of Sketch/SD only if the BS receives exact partial aggregate results that make up a good portion of the final result. This requires the BS to be fed directly by some sub-trees which cover a large part of the network. However, with strict restriction of topological correctness, such scenarios are unlikely to hold in the real world.

OPAG is a new data-aggregation scheme with zero computation error and good tolerance to moderate message losses. It separates in-network data aggregation into two layers: (1) at the data-aggregation layer, aggregation results are computed exactly along an overlay tree; and (2) at the underneath routing layer, a node opportunistically uses multi-path routing to send its partial result to a data-aggregation node (DAN). Also, the multi-path routing in OPAG differs from that of Sketch, SD and TD.

GRAB [43] proposes a credit-based forwarding scheme to control the redundancy of multiple paths between a data source and a data sink. Each sensor node is assigned a cost, which is the minimum energy to forward a packet from this node to the sink. Every packet is broadcast with a credit, and only the neighbors whose costs are below the credit re-send the packet. Therefore, the amount of credit determines the redundancy of the forwarding mesh. In contrast, OPAG is motivated to control the redundancy of data entries rather than that of the paths. We analyze the relationship between the data redundancy and the success ratio, and then design an algorithm that probabilistically forwards the data entries with minimum redundancy while retaining a given success ratio.

Boulis *et al*. [44] study the tradeoff between the aggregation accuracy and the energy consumption by taking advantage of the spatial-temporal correlation among sensor values. The idea is to create a system-level energy vs. accuracy knob whereby the less accurate the aggregation results, the less sensor values are used to estimate the aggregate results by using the data correlation more aggressively, and therefore less messages need to be exchanged. This approach is orthogonal to OPAG, as OPAG takes advantage of good network connectivity rather than data correlation. A

proper combination of the two approaches may achieve better accuracy-energy tradeoff.

In the context of splitting a packet over a number of disjointed paths with forward error correcting (FEC) codes, Dulman *et al*. [45] analyzed the relationship between the number of successfully delivering paths and the overall success probability of restoring the packet. By contrast, OPAG delivers each data entry via a set of paths without splitting it, and its analysis focuses on the relationship between the data redundancy and the success ratio of data delivery.

Parametric Probabilistic Routing [46] is a multi-path routing scheme for one-to-one communication in WSNs. Instead of specifying the paths between a source and a sink, the scheme allows each neighbor to forward packets with a certain probability based on the hop counts between the source, the sink, and the forwarding node. OPAG deals with in-network data aggregation, and the forwarding probability of a data entry is based on the data redundancy and the given threshold of success ratio.

Directed Diffusion [47] proposes a data-driven communication paradigm for WSNs. A data sink distributes a sensing task in the sensor network as an "interest," and the distribution process sets up gradients which are used to forward data from the data sources to the sink along multiple paths. Ganesan *et al*. [48] addresses the DD's drawback on energy-efficiency—DD requires periodic flooding to notify the sink and other nodes of available paths, and proposes a localized algorithm for setting up and maintaining alternative paths in advance. It also studies the tradeoff between the maintenance overhead of alternative paths and the resilience to node failures. These two schemes are not tailored to in-network data aggregation, and do not explore the relationship between the data redundancy and the success ratio, either.

Dozer [36] presents a data-gathering system which achieves impressive power efficiency—in the magnitude of 0.2% radio duty cycles. It uses a spanning tree topology and coordinates the communication between a parent and its children nodes by a TDMA protocol. Because wireless communication is inherently lossy. OPAG can complement this approach in reducing the number of retransmissions and lowering the energy consumption further.

2.7 Concluding Remarks

In this chapter, we presented a novel approach, called *Opportunistic Data Aggregation* (OPAG), for in-network data aggregation with no computation error and tolerance to moderate message losses in wireless sensor networks. By space-multiplexing messages, OPAG divides in-network data aggregation into two layers: (1) at the data-aggregation layer, intermediate aggregation results are computed exactly along an overlay tree; and (2) at the underneath routing layer, a node may send intermediate results to its aggregation node via multiple paths.

OPAG opportunistically uses multi-path routing to combat communication losses and achieve better energy-efficiency than using retransmissions. This is based on the observation that, when sending a message, the radio (e.g., the widely-used CC2420 radio) may consume much more energy in idle listening during the backoff period and the time in waiting for the acknowledgment than transmitting data bytes. Retransmitting a message is not energy-efficient because it incurs more idle listening on more backoffs and more time of waiting for an acknowledgment. In order to avoid extra idle listening, OPAG uses a multi-path routing scheme that differs from the traditional multi-path routing.

In order to use message space efficiently, OPAG uses a data-selection scheme that leads to optimal redundancy on high-priority partial results and provides as much message space as possible for low-priority results. The algorithm is based on an analysis of the relationship between the data redundancy and the success ratio over multiple disjoint paths.

We implemented OPAG on TinyOS-2.x and the TMote Sky node, and evaluated its performance on the Motelab Testbed, and TOSSIM. Experimental and simulation results show that OPAG performs much better than TAG and Sketch/SD under relatively good network connectivity, and the data-selection scheme provides improved performance when contention for the message space is moderate.

CHAPTER 3

Post-Deployment Performance Debugging

3.1 Introduction

Those who have been developing wireless sensor networks (WSNs) may sometimes observe poor application performance that was not seen in pre-deployment simulation or laboratory experiments. Such performance degradation may be attributed to several factors, such as different radio channel conditions, changed network topologies, undetected program bugs exposed in different execution environments, and the sensor instability caused by a harsh physical environment (e.g., temperature, humidity and vibration).

Post-deployment performance debugging is, therefore, very important, but often more difficult and more expensive than pre-deployment debugging in a laboratory setting. When a WSN application exhibits poor post-deployment performance, users (i.e., programmers and administrators) usually do not know which nodes in the network, nor which pieces of debugging information, should be examined closely to determine the causes of the problems. A common way of handling this problem, such as Sympathy [14] and MintRoute [15], is to make every node record detailed debugging information and send it to a data-sink node which then analyzes the information collected from all nodes. However, this approach may collect a large amount of information irrelevant to the observed problems, wasting battery energy on the sensor nodes and hence shortening their lifetime. It may also cause distraction from the effort of locating the problem sources, and interfere with execution of the underlying applications. Other approaches, such as NMS (Network Management System) [16], require users to query individual nodes interactively and iteratively. But, without any clue for the causes of the problems, they may have to query a large number of nodes before

locating and isolating the problems, thus taking a long time and consuming a significant amount of energy.

We argue that post-deployment performance debugging should focus on performance-in-the-large rather than the performance of individual nodes or even individual software modules for the following two reasons. First, the performance problems in a large WSN usually cannot be diagnosed by focusing on individual nodes or components. Second, post-deployment performance debugging is energy-expensive, and should, therefore, be turned on only when the overall application performance is unacceptably poor.

Our goal is to design, implement and evaluate a tool that helps users find the nodes and the software components that performance degradation can be ascribed to, and provides preliminary debugging information, such as the data loss rates and latencies on the nodes and components. According to the information, users can zoom in on these and nearby nodes, and extract more detailed debugging information, such as the neighbor table and the node state, to identify the causes of the problems using other debugging tools, like NMS [16], Sympathy [14], Dustminer [60] and PAD [61].

To achieve this goal, we propose a data-centric approach called *post-deployment performance debugging* (PDPD or PD2 for short). PD2 focuses on the data flows that a WSN application generates, and relates "poor performance" of the application to significant data losses or latencies of some data flows (i.e., problematic data flows) as they go through the software modules on individual nodes and through the network. Based on the data dependencies between different software modules and between different nodes, a few inference rules are derived for tracing back in each problematic flow. PD2 turns on performance monitoring of, and collects debugging information from, only those modules and nodes that the problematic flows go through. Then, PD2 visualizes the debugging information to locate the dominant sources (i.e., some software components on certain nodes) of significant data losses and latencies on the data flows, and hence, helps users locate performance problems.

PD2 can be applied to a wide range of WSN applications, in which users can express their expectation of the application performance with respect to message loss and/or latency of data delivery, and data routing paths (i.e., data flows) are relatively stable, such as data collection/aggregation, and event-driven applications with mobile data sinks.

PD2 is an automatic, efficient, and effective debugging tool for WSNs:

- Automatic: PD2 requires very small changes to application source codes and minimal human intervention in collecting the debugging information;
- Efficient: PD2 incurs minimal overhead when the application does not exhibit any performance problem, and avoids collection of irrelevant information and condenses redundant debugging information when it does;
- Effective: PD2 helps isolate the nodes and the software modules on these nodes where the problems occur.

The rest of this chapter is organized as follows. Section 3.2 describes the system model, the performance metrics of interest, and the goals of this work. Section 3.3 presents an overview of PD2, and Section 3.4 provides its detailed architecture. Section 3.5 experimentally evaluates PD2's performance on a real testbed. Section 3.6 discusses the generalizability and limitation of our approach. Section 3.7 discusses related work. Finally, we discuss future work and conclude the paper in Section 3.8.

3.2 System Model, Metrics, and Goal

This section describes the system model and performance metrics we will use, along with the goals of PD2.

3.2.1 Data-Centric Model

We model each WSN application as a directed graph, in which a node corresponds to a software component within a sensor node, and an edge indicates a potential data flow from one component to another, where the two components may reside in the same sensor node or two different sensor nodes. The application generates some data flows, each running on a *causal path*—a sequence of nodes traversed—in the graph.

For example, TestCollection, a WSN application in the TinyOS-2.x source, runs the Collection Tree Protocol (CTP) [37], in which the network forms a spanning tree rooted at the data sink, and each of the other nodes periodically sends its data to the sink via its parent. Figure 3.2 illustrates

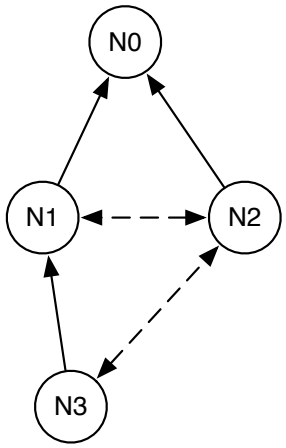


Figure 3.1: Each circle represents a node. A dashed edge from N1 to N2 indicates that N2 can hear N1. A solid edge from N1 to N0 indicates that N1 selects N0 as its parent node in the Collection Tree Protocol (CTP). N0 is the data sink.

the graph of TestCollection running on the network topology shown in Figure 3.1. This example assumes the TelosB mote platform with the CC2420 radio [49], but the model is applicable to any other platform. On each node, the sensor software is divided into three layers. At the top, the application layer has a component, TestCollection. In the middle, the network layer has six components: CtpForwardingEngine, CtpQueueSend, CtpRoutingEngine, LinkEstimator, AMSender, and AMReceiver. At the bottom, the MAC layer has one component, CC2420ActiveMessage.

TestCollection generates two types of data flows: data and beacon. The thick dotted curve represents the *causal path* for a data message from N3 to N0. In Figure 3.2, TestCollection on N3 initiates a data flow, and calls the send function provided by CtpForwardingEngine on the same node. CtpForwardingEngine sets the destination address to the parent node ID, and passes it to CtpQueueSend.[1] CtpQueueSend may buffer a number of messages, and send them one at a time by calling the send function of AMSender. AMSender, as a multiplexed sender, calls the send function of CC2420ActiveMessage to send it via the air. Upon receiving the message, CC2420ActiveMessage on N1 notifies CtpForwardingEngine through AMReceiver, a multiplexed interface. Likewise, CtpForwardingEngine continues to forward it to N1's parent, N0. When the

[1]The original implementation of CTP in TinyOS-2.x embeds a send queue into CtpForwardingEngine. We separate the two for finer modularization and debugging, which will be discussed later.

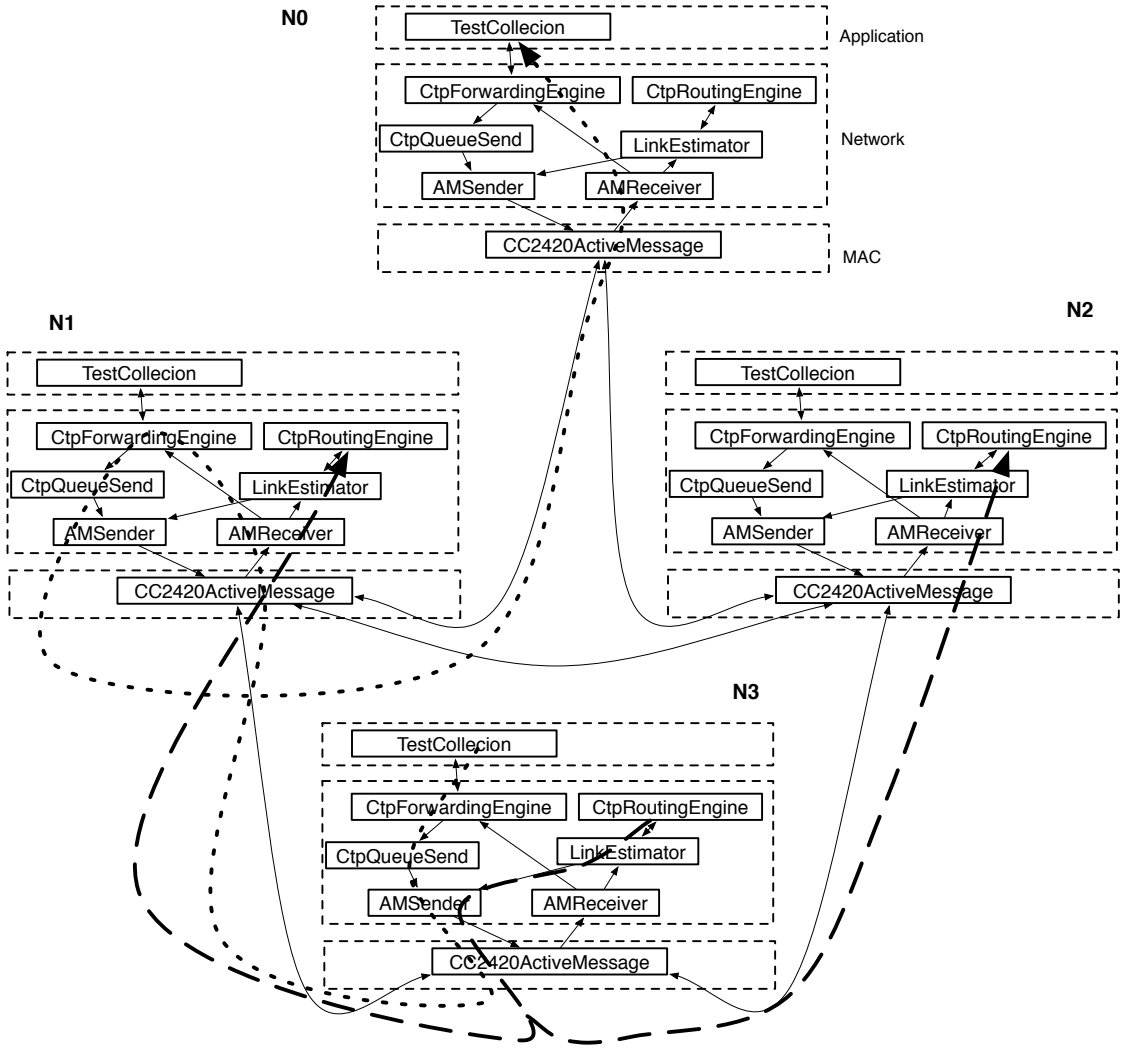


Figure 3.2: Each solid square represents a software component, and each directed edge indicates a possible data flow from one component to another. Each solid oval represents a virtual node of the radio channel. CC2420 radio is assumed, and the entire radio stack is considered as one component. A thick dotted curve represents the causal path of a data message, and a thick dashed curve represents the causal path of a beacon message. This figure only shows one data flow (from N3 to N0 via N1) and one beacon flow (from N3 to N1 and N2), and omits the other data/beacon flows for clarity.

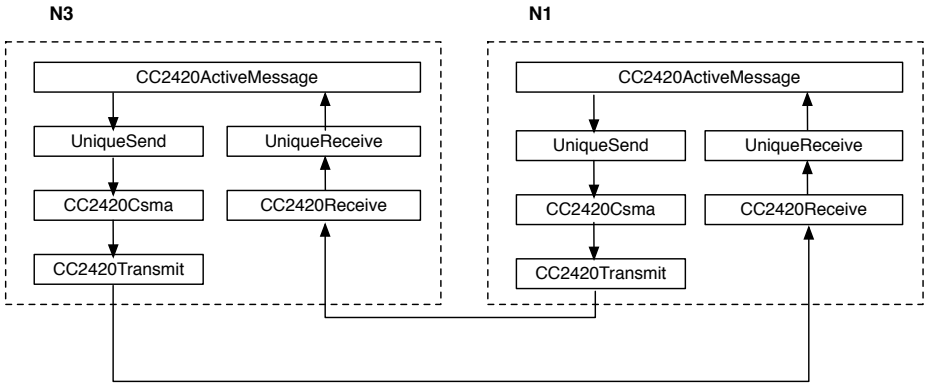


Figure 3.3: We assume low-power listening is disabled, and omit a few optional modules, such as CC2420AckLpl, PacketLink and CC2420TinyosNetwork. The thick dotted and thick dashed curves in Figure 3.2 show how a message traverses the radio stack.

data message reaches CtpForwardingEngine of N0, it is passed up to TestCollection because N0 is the sink.

The thick dashed curve in Figure 3.2 represents the causal path for a beacon message from N3 to N1 and N2. The CtpRoutingEngine component initiates the task of broadcasting a beacon message, and calls the send function provided by the LinkEstimator. LinkEstimator passes it to AMSender, which then sends it out through CC2420ActiveMessage. When receiving a beacon message, CC2420ActiveMessage of N1 (and N2) passes it up to AMReceiver, LinkEstimator, and finally, to CtpRoutingEngine. Beacon messages are used to estimate and exchange the link-quality information, which CtpRoutingEngine uses to choose the parent in the data-collection tree.

In our model, a "component" is a target entity of performance debugging, and may consist of one or multiple TinyOS nesC modules [50]. Whether or not to group several correlated modules into one component depends on the need of performance debugging. For example, the CC2420 radio stack consists of multiple nesC modules. If we need not look into the internal details of MAC, we can treat the entire radio stack as one component, CC2420ActiveMessage; otherwise, we decompose it into six components: CC2420ActiveMessage, UniqueSend, CC2420Csma, CC2420Transmit, UniqueReceive, and CC2420Receive, as shown in Figure 3.3.

3.2.2 Performance Metrics

Performance evaluation is subject to the nature of the application under debugging. The presentation of our approach confines its scope to data-collection applications in which each node periodically sends data to the sink. In Section 3.6, we will discuss how our approach can be applied to other applications.

We consider the sequence of data each node sends to the sink during a time-window as a data flow, and evaluate two performance metrics: the loss rate and latency of each data flow. The loss rate is the fraction of data that were sent but not received by the sink, i.e., lost or dropped along the corresponding causal path. The latency is the average delay from the time of data production to the time of its successful reception by the sink. Note that the latency is closely related to the loss rate. For example, radio retransmission can reduce the loss rate at the expense of longer latency.

We assume that the loss rate and latency of each data flow can be ascribed to the traversal of nodes along the causal path. More importantly, locating the traversal of nodes that contributes significant data loss and latency can help isolate the causes of performance problems. Note that the same node may incur different loss rates or latencies. For example, the radio component, CC2420ActiveMessage, is likely to incur higher loss rates and shorter latencies to the messages allowing only one retransmission than those allowing three retransmissions.

3.2.3 Goals of PD2

Our goal is to create a debugging tool and methodology that helps locate performance problems in a WSN application. Specifically, this tool should, in the context of our model,

G1. identify the causal path patterns—the repeated causal paths that account for significant data losses, and high latencies;

G2. locate the nodes and components on the causal paths that contribute to significant losses and high latencies; and

G3. deal with the resource constraints and the scalability issues of sensor networks.

Our tool is *not* for automating performance diagnosis, but for locating the causes of poor performance with the debugging information presented to us.

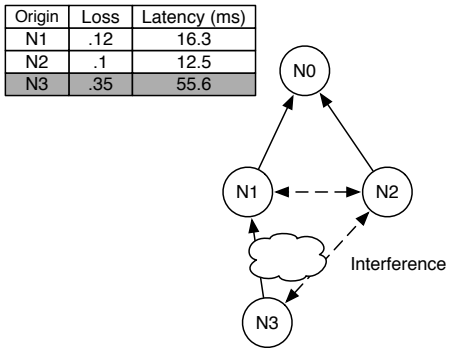


Figure 3.4: Node 3 suffers from radio interference, and consequently, its average loss rate and latency seen at the sink, N0, are considerably higher than those of other nodes.

3.3 Overview of PD2

How can a tool locate performance problems in a sensor network application without the global knowledge of the network? Our approach models the data flows generated by the application as node traversals in a directed graph of software components, and uses the data dependencies between components to trace back the causal paths (i.e. data flows) incurring high data losses or latencies. The performance of the components on the causal paths are monitored and reported to the sink. Then, the collected debugging information is visualized to help users diagnose the problems or examine closely the components with high losses and latencies using other tools.

Figure 3.4 shows an example that the application, TestCollection, exhibits poor performance because N3 suffers from radio interference. For bookkeeping performance, N0 computes the average loss rate and latency for each of the other nodes in the network. When it sees N3's loss rate is above a user-defined threshold 0.2, "poor performance" is alerted, and then PD2 is automatically invoked: (1) starting from TestCollection on N0, it traces back the causal path to TestCollection on N3 (shown as the left graph in Figure 3.5); (2) on each node in the reverse direction of a causal path, it turns on the debugging—letting the corresponding component record the loss rate and latency as the data goes through the component in both send and receive directions (shown as the right graph in Figure 3.5). PD2 periodically sends the average loss rates and latencies to the data sink, and helps us identify the communication from N3 to N1 as the dominant source of the loss and latency (indicated by the dark grey boxes). After a pre-specified duration, all components turn

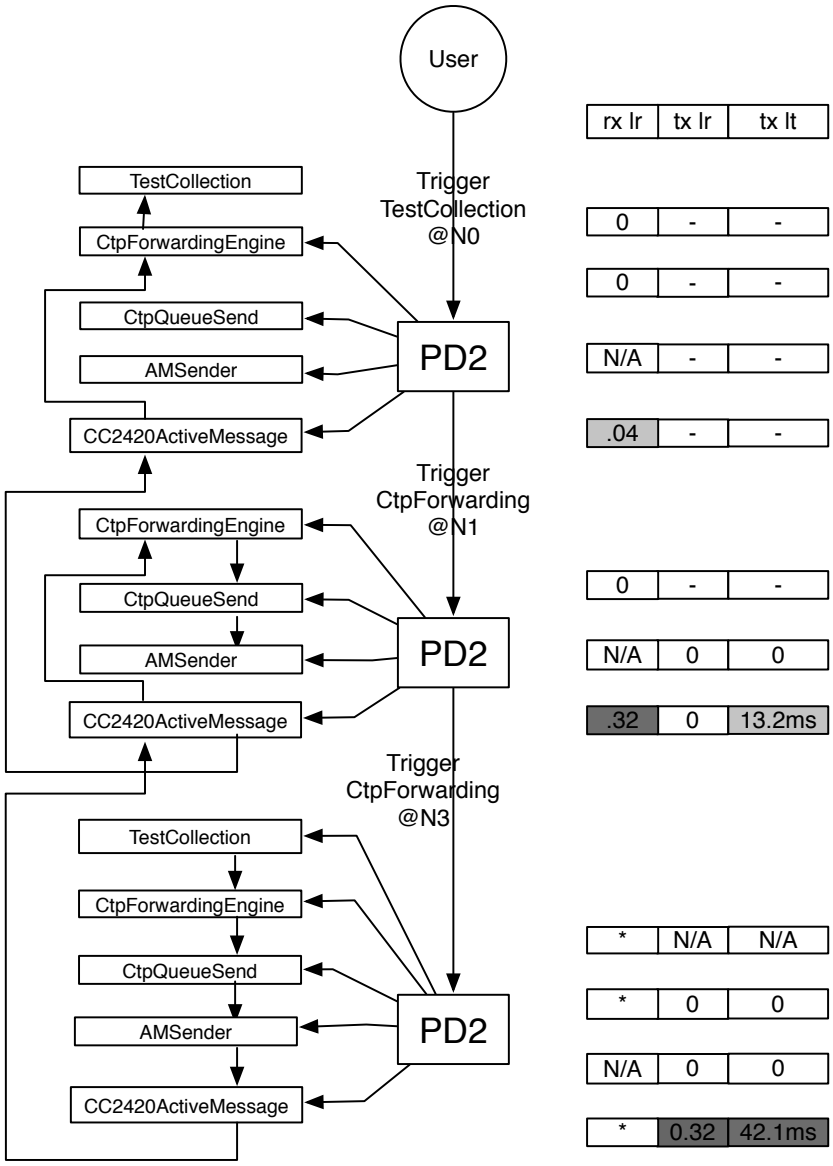


Figure 3.5: This figure illustrated how to trace back the causal path, and how to record the average loss rate and latency in both down and up directions. *rx lr*, *tx lr*, and *tx lt* represent the receive loss rate, the send loss rate, and send latency respectively. The receive latencies on all components are zero, and thus omitted.

off debugging, hoping that enough debugging information has been collected. If poor performance persists and the source has not yet been isolated, users may allow PD2 to turn on debugging again. Figure 3.5 shows the CC2420 radio stack as one component. However, if users need to analyze

what contributes most to the latency of the MAC, it is decomposed into several components, and more detailed information is thus collected.

Our approach consists of the following three phases:

P1. Define the debugging components and the causal relationship between the components manually or through a compiler. In this phase, we (the programmers and administrators) need to decide which nesC modules should be debugged, and define the components as a module or a group of several related modules. Then, we run a script which adds a few lines of simple instrumentation code into the corresponding modules, so that PD2 can record, at runtime, the loss rates and latencies when needed.

P2. Turn on debugging by tracing back the causal paths when poor performance is alerted. After deployment, the application performance is monitored at the sink node. If poor performance is alerted, i.e. above a user-defined threshold, PD2 traverses the casual paths in the reverse direction, and activates debugging on the components (modules) along the paths. The activation also specifies the frequency that the average loss rates and latencies are computed and reported to the sink, and the time when debugging is turned off. As the debugging data converge towards the sink, they may be condensed to reduce traffic load.

P3. Visualize the collected debugging information. As the debugging information comes in, it is visualized on the sink—we assume the sink has enough resources like a PC has. If we can isolate the modules/nodes that cause poor performance, we may sometimes infer the problem causes ourselves or by using a decision tree as Sympathy [14]. Or, we can use NMS to query specific information on the suspicious modules/nodes. Otherwise, we may have to reiterate the debugging process, refining the source modules and the debugging components, and repeating the three phases.

3.4 Architecture

When a WSN application exhibits poor performance, the sink initiates performance debugging by sending a triggering message to the Performance Debugging Communication (PDC) module, which triggers the Performance Debugging Engine (PDE) to turn on performance monitoring on

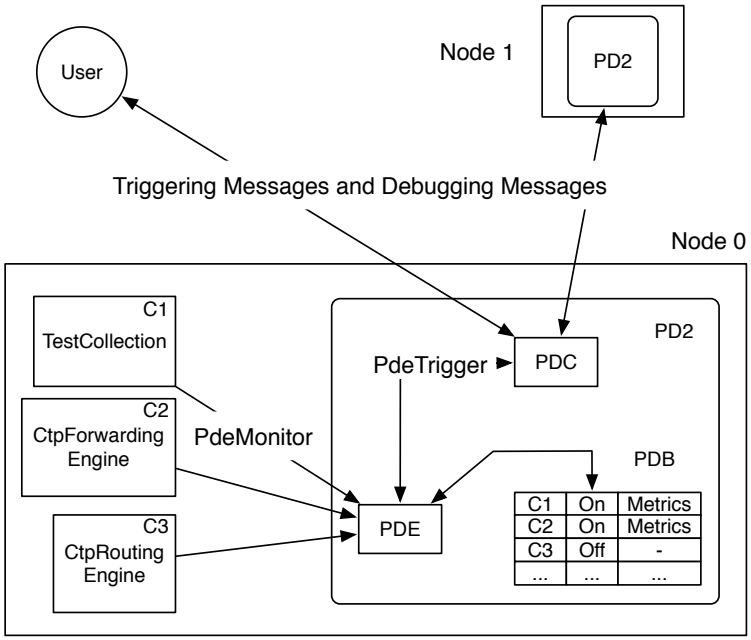


Figure 3.6: The architecture of PD2

the software component that is the endpoint of the causal path. PDE iteratively infers the previous component(s) on the path by using a set of triggering rules. If the previous component resides at a different node(s), PDE sends a triggering message via PDC to the node(s). With debugging of the software components on the causal paths turned on, PDE monitors the loss rates and latencies as data passes through the components, and stores them in the Performance Debugging Buffer (PDB). Then, PDE periodically sends, via PDC, the performance information to the sink. Figure 3.6 depicts the architecture of PD2.

3.4.1 Derivation of Triggering Rules

Triggering rules are derived from the causal relationships between components. Typically, if traversing component C1 causes the traversal of component C2, the corresponding rule specifies that turning on the debugging of C2 triggers the debugging of C1. In order to systematically derive the triggering rules, we classify the casual relationships as follows.

- **Pass-Down**: An upper-layer component passes the data to a lower-layer component. For example, TestCollection calls the *send* command of CtpForwardingEngine.

- **Pass-Up**: A lower-layer component hands the data off to an upper-layer component. For example, CtpForwardingEngine signals the *received* event of TestCollection.
- **Send-Receive**: A component of one node sends out the data to a component of another node. For example, a data message sent by CC2420ActiveMessage of N3 is received by CC2420ActiveMessage of N1.
- **User-Defined**: This class of causal relationship is entirely up to users. However, users should be prudent to avoid arbitrary interpretation of causality between components.

The relationships, Pass-Down and Pass-Up, are easy to identify, because we only need to look for the components connected by Send, AMSender, and Receive interfaces. Send-Receive is always applied to the communicating components on different nodes. For example, if we treat the CC2420 radio stack as one component, they both are a CC2420ActiveMessage; if we are interested in the internal workings of the CC2420 radio stack, the CC2420Receive on one node may trigger the CC2420Transmit on another node.

3.4.2 Performance Debugging Engine

At the core of PD2 is the Performance Debugging Engine (PDE) which provides two interfaces: PdeTrigger and PdeMonitor. The former is used for interaction between PDE and PDC, while the latter is used to monitor the performance of software components.

The most important function of PDE is to infer the previous component(s) on the causal path. Upon receiving a triggering message from the sink or the PDC of another node, PDC turns on performance monitoring on the component as specified in the message. Then, PDE iteratively applies a set of triggering rules to infer the next component to be triggered, i.e., the previous component on the path. If the next component(s) resides at a different node(s), PDE signals an event to PDC, which then broadcasts triggering messages to notify the component of those neighbors on the data path. A data path may be labeled along the forwarding nodes by saving the origin id or using a blooming-filter[52].

Suppose N3 suffers from significant data loss. Then, the sink triggers debugging on the component, TestCollection on N0 with the dependency, Pass-Up which indicates incoming data with

regards to N0. By iteratively applying the appropriate rules, PDE triggers the components, CtpForwardingEngine, and CC2420ActiveMessage (AMReceiver is a trivial interface component, and thus omitted). When CC2420ActiveMessage is triggered, PDE changes the data dependency to Send-Receive, because the data must be from a neighbor(s). Then PDC broadcasts a message to trigger CC2420ActiveMessage of its neighbors. Note that not all neighbors send N3's data to N0, i.e., not all neighbors are on the causal path. Each neighbor (N1 or N2) checks if it has sent N3's data by a sent-cache or a bloom-filter bitmap which records the IDs of the origins of the data it has forwarded. N2 ignores the triggering request because it is not on the causal path.

CC2420ActiveMessage on N1 is triggered with the dependency Pass-Down which means outgoing data with regards to N1. Then, PDE of N1 triggers the component, AMSender, CtpQueueSend, and CtpForwardingEngine using the rules. As a forwarding node, N1 changes the incoming data from N3 to outgoing at CtpForwardingEngine. Accordingly, PDE has a user-defined rule to change the data dependency from Pass-Down to Pass-Up. By Pass-Up, CC2420ActiveMessage of N1 is then triggered.

Similarly, N1 broadcasts a triggering message to its neighbors. N0 just ignores the message because it is a duplicate triggering request. PDE of N3 then triggers the component, AMSender, CtpQueueSend, CtpForwardingEngine, and finally, TestCollection, which is the starting point of the data path.

As data goes through a component with debugging turned on, it signals an event to report its action on the data to PDE. Then, PDE stores the debugging information in PDB, which has an entry for each component for each flow. PDE sends the debugging information every period (specified in those triggering messages). Debugging is turned off after a specified duration, and only the components on the causal paths report debugging information. This reduces the energy cost of, and the distraction by, irrelevant information.

3.4.3 Grey-box Performance Monitoring

We consider each debugging component as a grey-box: we only record the loss rate and the latency that the module imposes, and ignore the internal workings; if the internal states are needed for diagnosis, we decompose the component into several finer subcomponents.

Three operations are monitored: *send*, *sendDone*, and *receive*. A message is considered suc-

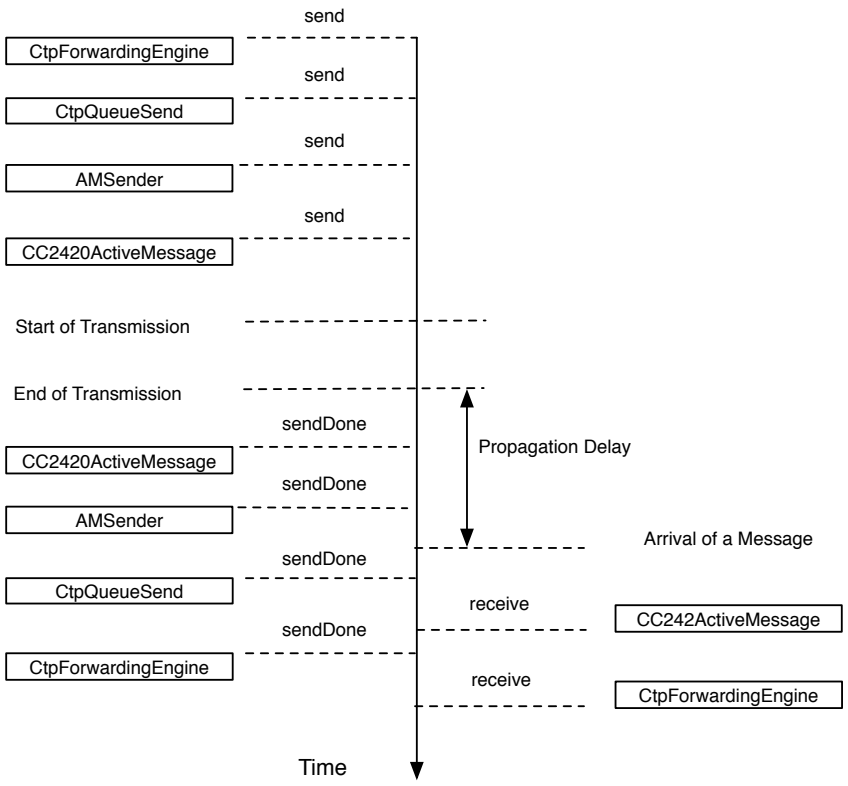


Figure 3.7: Monitoring of latency

cessfully sent if the command *send* is executed successfully and the event *sendDone* is notified successfully. So, the send loss rate of a component is one minus the number of successful sends over the total number of sends. On the receiving side, the receive loss rate of a component is one minus the number of receive events over the total number of messages sent out (calculated from the message sequence numbers).

Simple instrumentation codes are inserted by a preprocessing script, which goes through the configuration files that wire different components. If a component in a configuration file is labeled to be instrumented, the script inserts the instrumentation code to the appropriate locations in the corresponding module file.

Figure 3.7 shows how the latency is monitored. Basically, the send latency of a component is the time between send and sendDone minus the latency of its lower component. For example, the latency incurred by CtpForwardingEngine is the time between CtpForwardingEngine's send and senDone minus the time between CtpQueueSend's send and sendDone.

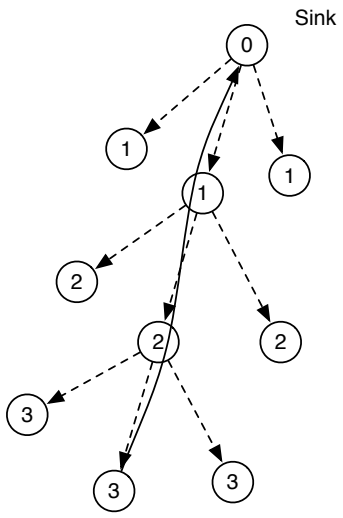


Figure 3.8: Each circle represents a node. The solid curve indicates the data flow, and the dotted arrows indicate the propagation of triggering messages. The number enclosed in each circle is the gradient of the node.

3.4.4 Communication of Performance Debugging Information

PD2 does not assume any routing scheme, because routing itself may cause performance problems. When propagating the triggering messages, PDC simply broadcasts them to all neighbors; each neighbor decides whether or not to continue propagation of the triggering messages based on if they are on the causal paths.

Sending debugging messages to the sink is more complicated. We need a reliable and simple routing scheme that incurs low data loss and low complexity. Flooding is an option, and may perhaps become a good option when the network suffers from significant message losses. The shortcoming of flooding is that it generates lots of traffic and consumes a significant amount of energy.

We use the simplest gradient-based, multi-path routing. As triggering messages propagate along the reverse direction of casual path, each node finds the hop-count distance between itself and the sink, as shown in Figure 3.8. The hop count is used as the gradient of that node. Each node broadcasts its debugging messages to all neighbors, and only those neighbors with smaller gradients re-broadcast the debugging messages.

3.4.5 Aggregation of Debugging Messages

The goal of aggregating debugging messages is to condense uninteresting debugging information, while extracting and preserving useful information. If none of the components on a node contributes to significant data losses or latencies, we can suppress their debugging information into a more succinct "OK" message. Moreover, a relay node can merge multiple OK messages to further reduce the amount of uninteresting debugging traffic. In order to aggregate as many OK messages as possible, the nodes closer to the sink wait for a certain amount of time before sending out the merged OK message.

3.4.6 Handling Incomplete Debugging Information

What if the collected debugging information is incomplete due to radio loss? In the worst case, what if the radio is so jammed that some nodes cannot report any debugging information? There are two ways to deal with incomplete debugging information: visualization and delayed reporting.

We visualize both useful debugging messages and OK messages. If certain nodes report much less debugging information than others, we label them with lighter colors. We should then see void areas or holes in the network. With this knowledge, we can switch to flooding, or use other debugging tools, such as NMS[16], to query key information of specific nodes, e.g. the neighbor table, the radio state, and the average backoff time. If these nodes are not reachable, we will be able to see the changes of network topology to avoid the jammed areas.

Delayed reporting assumes that severe communication loss is transient. The nodes store the debugging information in flash memory, and attempt to send them when the network condition improves.

3.4.7 Visualization of Results

We first filter out duplicate information from the debugging information delivered to the sink, and then use graphical tools to visualize it.

On top of a network map, we plot the components next to each sensor node, then draw edges between the components based on the causal relationship contained in the debugging information. Then, we label the performance metrics next to each component. An example figure is shown

in Figure 3.9. CtpForwardingEngine of Node 1 sends about 17 messages because TestCollection of Node 1 (not shown in the figure) generates 10 messages. The dotted entries do not show significant loss rates or latencies, and therefore they are not sent to the sink in order to reduce debugging overhead.

3.5 Evaluation

We implemented PD2 on the TMote Sky platform [33] and evaluated it on Motelab [32] at Harvard University. We first present implementation details. Then, we describe our evaluation methodology and experimental setup. Finally, we present the experimental results.

3.5.1 Implementation Details

We implemented PD2 on the TinyOS 2.0.1 and TMote Sky platform, which is equipped with an 8MHz TI MSP430 processor, 10K RAM, and a 250Kbps Chipcon radio operating at 2.4GHz. It supports various types of sensors, including light, temperature, and humidity sensors.

PD2 assigns an ID to each of the following components: TestCollection, CtpForwardingEngine, CtpRoutingEngine, CtpQueueSend, LinkEstimator, AMSender, CC2420ActiveMessage. Each component, except for CC2420ActiveMessage, has two entries, one for Pass-Down and the other for Pass-Up. Each entry records the total number of messages the corresponding component sends (Pass-Down) or receives (Pass-Up), the loss rate, and the latency of successful send or receive. CC2420ActiveMessage may store multiple entries, because it may receive or send (using unicast) messages to different nodes — in TestCollection, a node may receive data from multiple child nodes, and may send data to a different parent node that may change, depending on link quality. A buffer is allocated to store at most 30 entries. When the buffer is full and a new entry needs to be added, the entry with the least loss rate is replaced.

PD2 has a 30-byte bitmap to store the data-path information. That is, a node hashes into an index the origin ID of each message it forwards and sets the corresponding bit in the bitmap. Therefore, in the triggering process, PD2 can determine if a node is on the data paths by checking that bit. The original implementation of CtpForwardingEngine maintains a message queue so that it may buffer the messages from its child nodes. For finer debugging, we create a new component,

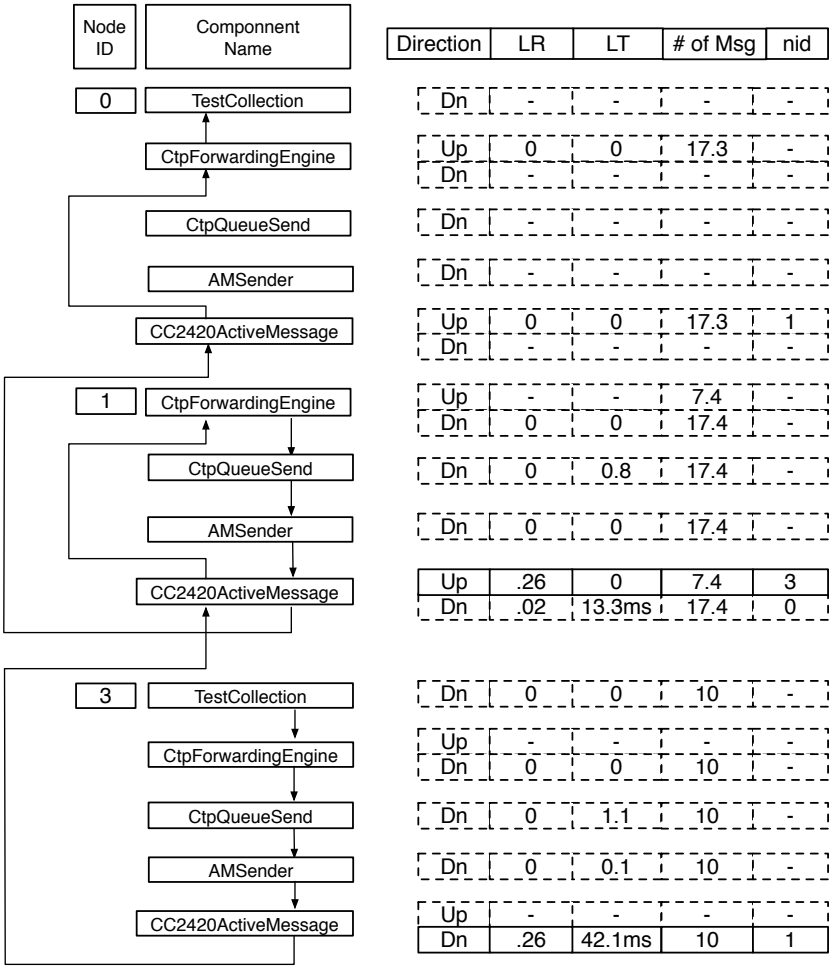


Figure 3.9: Visualization of debugging information. *LR* and *LT* indicate the loss rate and latency respectively. *# of messages* means the number of messages a component sends or receives. *nid* is the ID of the sender (incoming data) or receiver (outgoing data).

CtpQueueSend, which keeps the message queue and interacts with the lower-level communication component, AMSender. The "thiner" CtpForwardingEngine basically calls the send function of CtpQueueSend to send its own data or forward the data from a child node.

3.5.2 Evaluation Methodology

The effectiveness of PD2 hinges on the premise that it can accurately locate the "problem" node(s) and software component(s). To evaluate this premise, we inject into the network the following three types of sources of performance problems.

1. Code bug: after CtpQueueSend sends a message, it does not put the message buffer back into the message pool, and therefore runs out of message buffer quickly.

2. Weak link: a node uses a low transmission power so that it has a relatively weak link to its parent node.

3. Radio interference: a node creates radio interference by continuously sending messages.

We ran the data-collection application, TestCollection on Motelab, which has about 160 Tmote Sky nodes scattered across a number of rooms on three floors. The environment should have enough multi-path effects from obstacles and interferences from other wireless communications as in a real sensor network.

Each node uses a CC2420 radio operating at the default radio frequency—channel 11 of IEEE 802.15.4 (2.405 GHz) with maximum transmission power. It uses the CC2420 radio stack in TinyOS-2.0.1, which sends 802.15.4-compliant packets, but does not implement the 802.15.4 MAC protocol. The physical header and the MAC header use 20 bytes. The maximum message payload length is 50 bytes. In TestCollection, each node sends its data to the sink (node 2) every 30 seconds. The period for collecting debugging information is set to 5 minutes. The energy consumption is estimated by the number of messages transmitted,

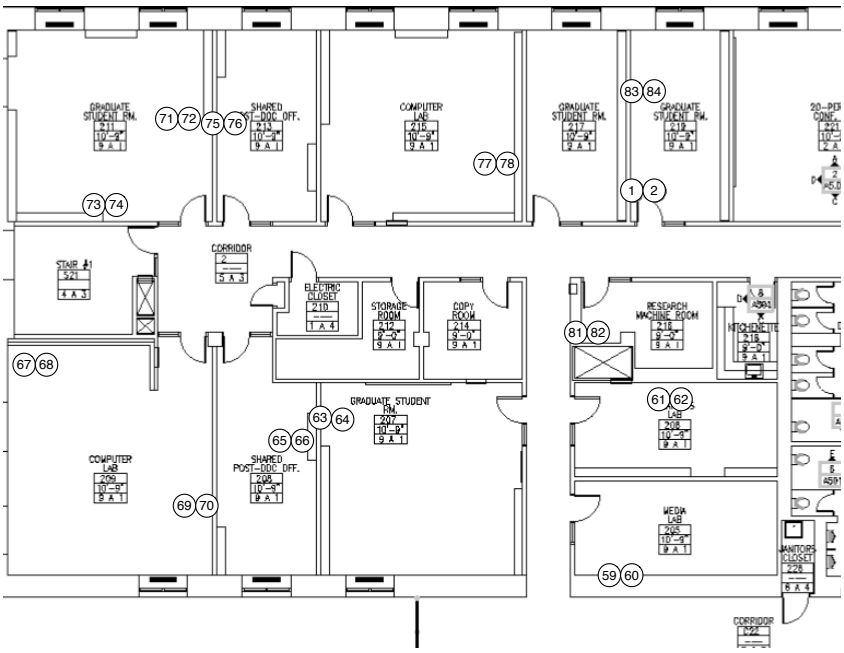


Figure 3.10: The left portion of the second floor in the building where the Motelab testbed is deployed.

3.5.3 Experimental Results

Three types of performance problems

We copy the map of the testbed from the testbed web site, and visualize the debugging information on top of the map. Figure 3.10 shows a portion of the testbed for the purpose of illustration.

Figure 3.11 shows the debugging information on top of the map (the room arrangement in the background is removed for better readability, but the network topology is retained), with the queue bug injected into node 63. The sink, node 2, finds that both node 72 and 68 exhibit relatively high loss rates, as indicated by the stars. PD2 traces one possible problem at a time, e.g., node 72. Node 72 has two paths to the sink due to change of its parent node, so both paths are traced. As a result, debugging of TestCollection, CtpForwardingEngine, AMSender, and CC2420ActiveMessage on node 2, 77, 63, and 71 is turned on, and they report the debugging information as shown in the figure. Node 71 and 77 only report OK messages, because none of the components on the two nodes recorded any significant loss. Node 63 reports one entry in the format of node ID, direction of the data (Up or Down), component ID, loss rate, latency, the number of messages,

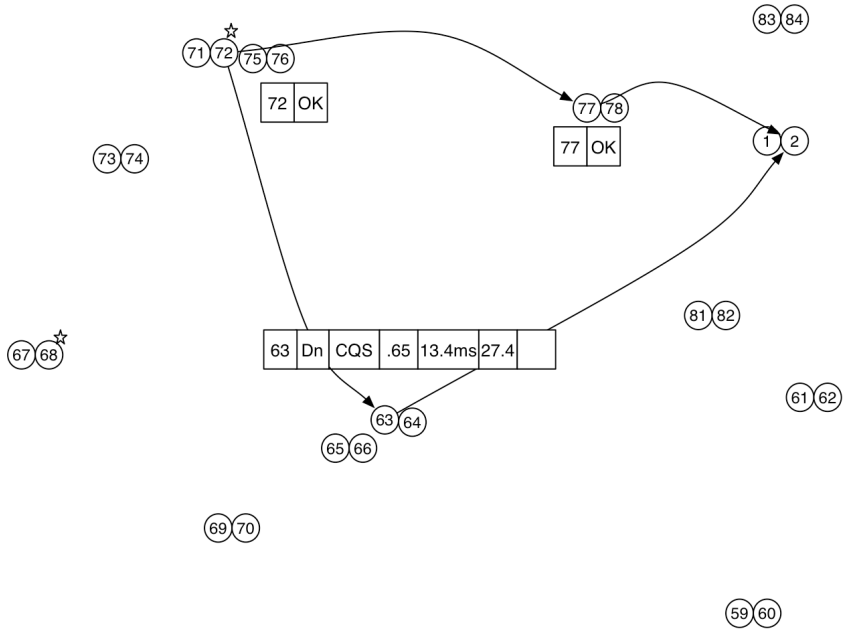


Figure 3.11: Node 63 with a queue bug injected cannot forward messages. However, its data can be sent without any problem.

and the ID of the sender (Up) or receiver (Down) (for CC2420ActiveMessage only). CtpQueueSend records a loss rate of 0.65, because only the forwarded messages need to be buffered in the message pool. The node's own data is created by TestCollection(TC), then passed down to the CtpForwardingEngine (CFE), CtpQueueSend (CQS), AMSender (AMS), and CC2420ActiveMessage (C2A) without causing any data loss. The other entries of node 63 are not reported because they do not record any significant loss.

Figure 3.12 visualizes the debugging information after lowering the transmission power of node 69. As a result, node 69 may select node 65 or 64 as its parent, and therefore its outgoing messages are split between the two possible parents. Both links suffer a loss rate of about 0.18. Due to retransmission within the MAC, the latency increases to about 2-3 times of the latency without retransmission. At each receiver, the debugging entry records the loss rate and the number of messages received. Node 81 and 61, albeit with debugging turned on, only report OK messages because they do not record any high loss rate.

Figure 3.13 visualizes the debugging information after node 64 generates radio interference. Several nodes may be affected, but we only show node 63 and 81 in the figure. The difference

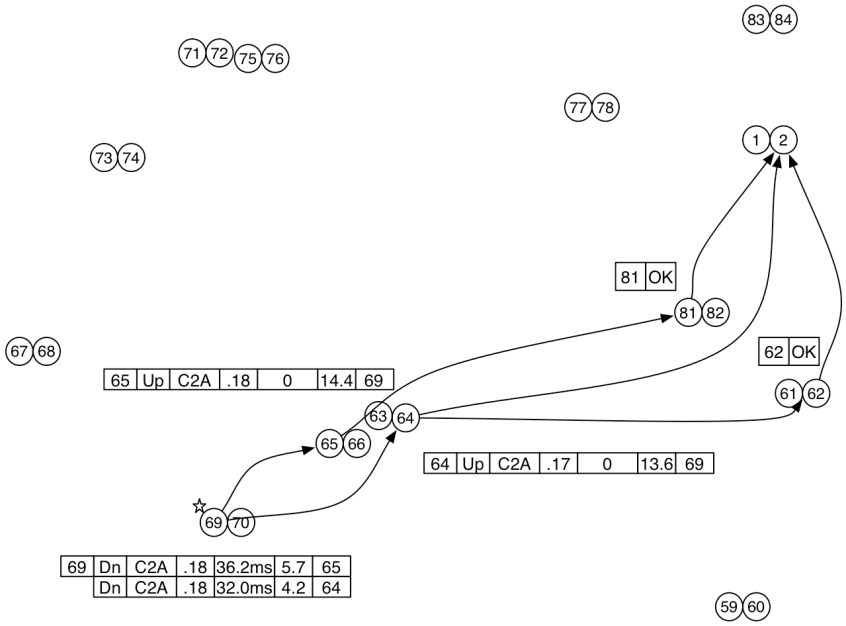


Figure 3.12: Node 69 uses a lower transmission power, and hence, its links to possible parents are relatively weak.

between radio interference and weak links is that the former leads to a longer backoff (waiting for a clear channel), and thus a longer latency. Although radio interference tends to affect more nodes than the previous two cases, we can still isolate an area of interference.

Overheads

Adding PD2 to a sensor network application incurs overheads in terms of extra network traffic and code size.

The additional network load incurred by PD2 consists of triggering and debugging messages. If the tree topology of the data-collection application changes over time, a data flow may take a few different data paths. If the source of a performance problem is far away from the sink, then the data paths are long. To evaluate how both factors affect the overhead of PD2, we run the data-collection application as follows. First, all nodes, after their boot-up, are given 3 minutes to warm up their neighbor table and build a tree topology. Then, we pick a node k ($k = 1,2,3,4$) hops away from the sink, and inject a queue bug into that node. The tree topology may change (Variable-Topo), or remain fixed (Fixed-Topo). The debugging messages are held for 1 second for

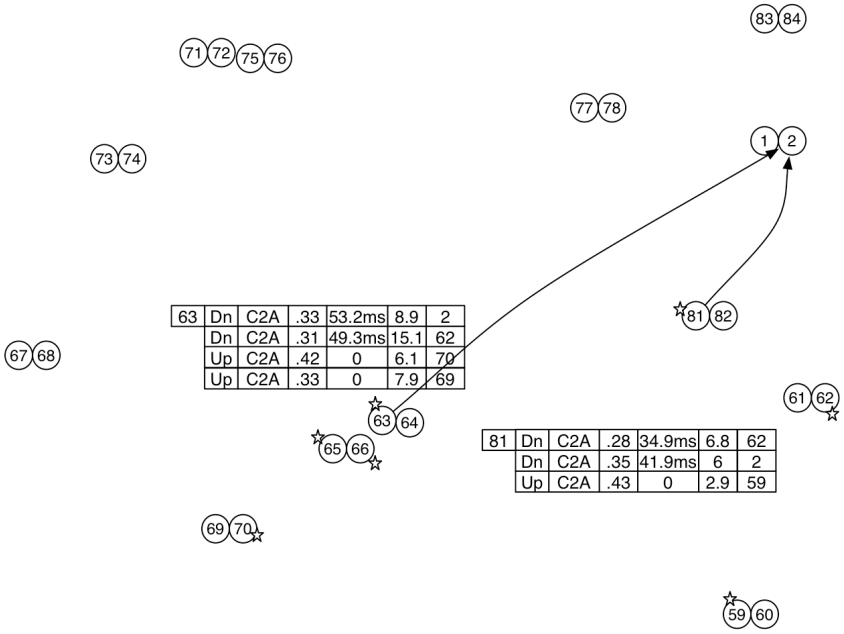


Figure 3.13: Node 64 creates radio interference by continuously sending messages.

possible aggregation. For comparison, we also ran tests which require all nodes to report debugging information to the sink. All communication overheads are normalized to that of Fixed-Topo with aggregation.

Figure 3.14 shows that PD2 incurs 5–10% of the overhead of collecting the debugging information from all nodes, depending on how far away the injected problem is from the sink. Changes of the tree topology increase the network overhead significantly, because data flows along a braided path rather than a single-threaded path, and more nodes need to report debugging information to the sink.

Figure 3.15 shows that, as each node waits longer, aggregation of more debugging messages is more likely. One second of waiting allows aggregation of the debugging messages within a few hops, and reduces the overhead by 35%. Five seconds of waiting does not save as much, because the message space limits the amount of debugging information that can be aggregated.

The network overhead also depends on the period of collecting debugging information. Since PD2 computes the averages of loss rate and latency during each period, the period should not be too long; otherwise, it may be difficult to capture transient performance problems. We find setting the debugging period as long as 10–20 data collection periods works well.

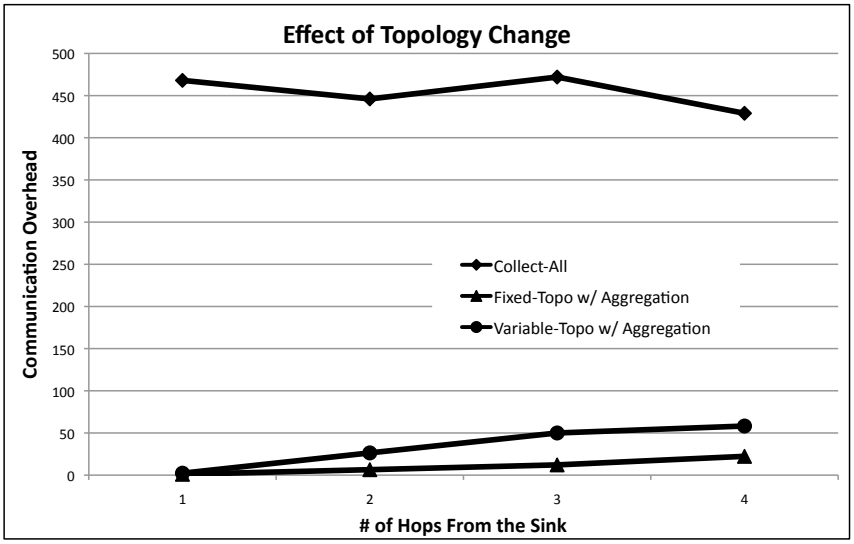


Figure 3.14: By selectively collecting debugging information, PD2 incurs about 10% or less of the overhead of collecting information from all nodes.

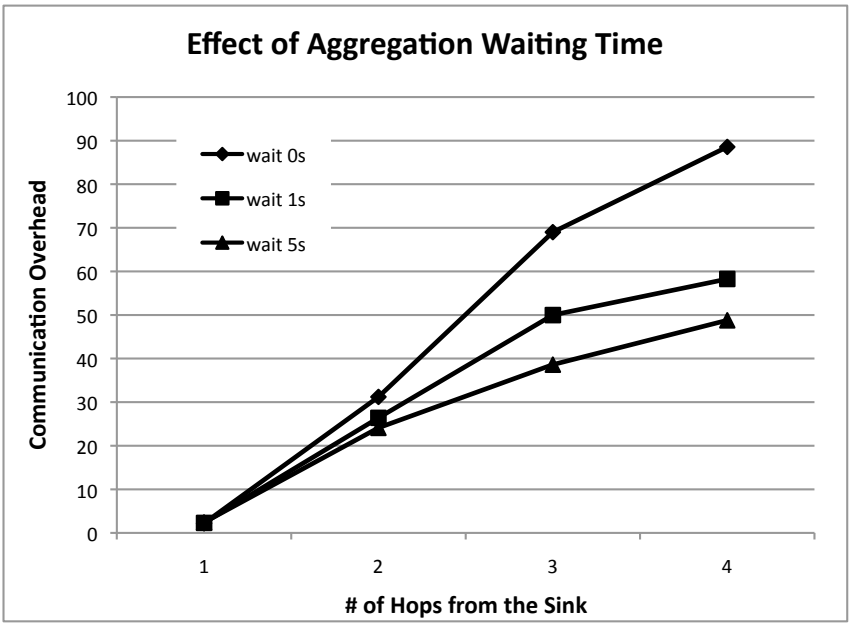


Figure 3.15: A node may wait some time before sending its debugging information. A longer wait allows aggregation of more debugging messages, thus reducing the network overhead further.

PD2 consumes about 800 bytes of RAM space and 600 bytes of code space. The RAM space is used for bookkeeping the time and return status of send, sendDone, and receive calls by different components, and storing the performance metrics. We argue that, by keeping per-component status and debugging information, PD2 is able to collect debugging information from selective nodes rather than the entire network, and trades memory usage for significant reduction of the communication overhead. With the recent-generation sensors, such as TMote (10KB RAM)[49] and iMote (64KB RAM)[53], memory is not as scarce as battery power. Hence, a slight increase of memory usage for energy savings (via reduction of communication overhead) is an acceptable tradeoff.

3.6 Generalizability

So far, we have considered a broad class of applications which collect data in WSNs based on the spanning tree routing. Furthermore, PD2 can be adapted to other applications in which users can express their expectation of the application performance with respect to message loss and/or latency of data delivery, and data routing paths (i.e., data flows) are relatively stable. In this section, we first exemplify the generalizability of PD2 through two important classes of applications: data aggregation and event-driven applications with mobile data sinks. Then, we discuss the limitation of PD2.

3.6.1 Data Aggregation

Data aggregation, an important category of sensor network applications, requires the base station to compute the aggregates (such as average and count) of sensor readings by a sensor network. Instead of directly collecting all sensor readings and computing the aggregation results at the base station, *in-network* aggregation approaches, such as TAG [9], Stretch [10] or SD [11] and OPAG [54] allow the sensor data to be aggregated as they are routed towards the base station. In-network data aggregation significantly reduces communication traffic and improves overall energy-efficiency [9].

The main performance metric of data aggregation is aggregation accuracy, which depends on how many data samples contribute to the aggregation results — the less communication loss, the more data samples are aggregated, and the more accurate aggregation results become. Therefore,

the application performance can be expressed by the loss of data samples at the base station.

When a user perceives the loss of data samples above a certain threshold (e.g., 20% of the total network), s/he can send a triggering message to the PD2 on the base station. Based on the send-receive dependency rule, the base station propagates the triggering message to all child nodes. If a child node suffers from loss of data samples above the threshold (e.g. 20% of the sub-tree rooted at the child node), it turns on debugging and continues to propagate the triggering message to its children; otherwise, it ignores the triggering message. In this way, the triggering message is propagated along those routing-aggregation paths (i.e., data flows) that suffer significant loss of data samples.

The main difference between data aggregation and data collection is how an intermediate node handles the data it receives. In data aggregation, it aggregates all the intermediate results from its child nodes and then sends the new intermediate results to its parent node, while, in data collection, it lets each data flow from a child node go through by simply forwarding the intermediate results of the child to its parent. However, because data dependency is retained as data flows are aggregated, PD2 can be applied to debug data aggregation applications.

3.6.2 Event-Driven Applications with Mobile Data Sinks

Many event-driven applications operate on a hybrid network of many stationary sensors and a few mobile sinks. Each sink may subscribe to certain interesting events and expect to receive notifications from the sensors which detect those events. The application performance is usually determined by the latency and loss rate of the notification messages.

Because the mobile sink moves around, the routing paths from a sensor source (which detects an event of interest) to the mobile may change continuously. Then, how can a triggering message, initiated by the mobile, trace back the routing paths (i.e. data flow) to the source?

Fortunately, the changing routing paths to the mobile sink usually share significant common parts. Some routing protocols for mobile data sinks, such as GPSR [17, 18], Landmark Routing [19], and Beacon Vector Routing (BVR) [20] exploit certain location service protocols, for example, GLS [21] and DLSP [55]. The basic idea is that a mobile sink periodically publishes its location (or virtual coordinate) to a set of sensor nodes, called *location servers* (which are elected according to some well-known rules). Notification messages are first routed to a nearby location

server by the rules, then to the known location of the mobile, which is updated to the server. Even as the mobile moves, the segment of the routing paths from the source to the location server do not change. Therefore, PD2 can be used to debug the latency and the message loss on that segment (the mobile can send a triggering message directly to the location server using user-defined dependency, and the location server can propagate the triggering message in the reverse direction along the routing path to the source).

If the mobile moves relatively slowly, the mobile does not update the location servers frequently, and the segment from the location server to the known location stored on the server does not change for a while. Then PD2 can also be used to debug the latency and loss of notification messages on that segment of the routing path. For optimization, a source may cache a location of the mobile within a timeout period, so the notification messages during the period need not go through the location server every time. Then PD2 can be used to debug the routing path from the source to the cached location during the period (in this case, the mobile sends a triggering message directly to the node closest to the location, which then propagates it back to the source).

If the mobile moves fast, it becomes difficult to identify and debug the segment from the location server to the mobile. A simple and effective way is probably to turn on debugging on all nodes. The performance monitoring and debugging information collection of PD2 are still applicable.

Other approaches, such as TTDD [56], establish a grid infrastructure to disseminate information of events. A mobile sink floods a request within a local grid to reach a data dissemination node. PD2 can be used to debug the data dissemination within the grid infrastructure.

3.6.3 Limitation

The main limitation of PD2 is its reliance on relatively stable data flows. If the data flows generated by an application change radically and frequently (e.g. when a mobile data sink moves very fast), PD2 cannot effectively identify the data flows and turn on debugging on the nodes along the data paths. Instead, it is better to turn on debugging on all nodes, because the data flows probably go through many nodes anyway.

The debugging information PD2 collects is preliminary. It does not provide the extensive debugging information that other tools collect, and therefore may not be directly helpful in identifying complicated problems, such as race condition or other problems that crash a node. However, it is

useful in isolating the problematic components and nodes so that users can zoom in and closely examine them using other debugging tools.

3.7 Related

The state-of-art of WSN debugging relies on simulation, monitoring & logging, and visualization. Programmers and well-known debugging tools, such as Sympathy, use all these techniques.

Simulation [51, 41, 57] is useful for shortening the development cycle, repeatable execution, and particularly, easy evaluation of various network deployment scenarios. But it cannot substitute debugging on real devices because it is usually not possible to accurately simulate the real-time dynamics of the channel condition and the physical environment, nor the hardware-related details, such as timing and radio activities. Monitoring and logging [14, 15, 16] are the basic means of debugging on real devices. Programmers must be prudent of where and what to monitor, but may still be obfuscated by excessive and irrelevant information. Visualization [58] is to graphically present the debugging information from simulation, or monitoring and logging on real devices, and help programmers understand the debugging information and locate the problems.

NMS [16] facilitates the monitoring of a WSN application by providing access to the counters and statistics instrumented in TinyOS nesC modules. To reduce the amount of debugging traffic, programmers query a node or a group of nodes through the data sink by using the dissemination protocol, and the node(s) may then read the counters, compute the statistics, and report them to the sink. Moreover, NMS lets each node log unexpected events in a persistent local storage (flash memory), and programmers can then issue commands to retrieve the logs on any specific node.

Sympathy [14] lets each node actively report a carefully-selected set of metrics to the data sink once every *metric period*. Given the debugging information collected at the sink, Sympathy detects failures and determines their causes by using a decision tree. MintRoute [15] also includes periodic reports of the neighbor table for debugging at the sink, but does not automate failure detection and cause analysis.

Declarative Tracepoints (DT) [59] introduces TraceSQL to program the debugging actions as tracepoints and allow addition and deletion of tracepoints at runtime. It provides a systematic framework where multiple debugging techniques can co-exist. Dustminer [60] and PAD [61] are

diagnosis tools that analyze the collected debugging information (events) to uncover causes of performance degradation.

Performance debugging for distributed systems of "black boxes" (PDB) [62] motivated our work, although it was proposed to deal with generic distributed systems consisting of "black-box" (closed source) components. PDB models application activities as node traversals in a graph of communicating components. Given a trace of messages, PDB proposes two algorithms to determine the causal relationships among messages, and then isolates the performance bottlenecks. PD2 proposes a similar model for WSN applications, but the causal relationship is determined by the compiler or defined by the programmer—it is known to PD2 because there are no black-box components. The focus of PD2 is to use the causal relationships to automatically trigger the relevant components on a selective set of nodes to log and report debugging information to the sink.

Visibility [63] proposed a different angle to debugging sensor network protocols: increasing the visibility of the network protocols in their designs. Visibility is defined as the energy cost of diagnosing the cause of a behavior in a protocol, and used as a quantitative metric to evaluate and compare protocols. By considering the new design metric, sensor network protocols can be made more debugging-friendly.

Interface contracts for TinyOS [64] proposed a static-checking approach to specifying and enforcing pre- and post-conditions on function calls provided by a component. Some bugs in TinyOS 1.x were actually caught by this tool.

Passive Distributed Assertions (PDA) [65] allows developers to assert certain properties of a distributed system, and the assertions are inserted into the program code like in C programs. To verify the assertions, the affected sensor nodes broadcast some status information, which is collected and analyzed through a separately-deployed sniffer network.

3.8 Concluding Remarks

In this chapter, we proposed a data-centric approach to post-deployment performance debugging (PD2) of WSN applications. Our goal is to help users locate where performance problems occur and provide useful hints for fixing them or closer examination.

PD2 focuses on the data flows that a WSN application generates, and models them as causal paths in a graph of software components. PD2 relates poor application performance to significant data losses or latencies of some data flows (problematic data flows) as they traverse the software components on individual nodes and through the network. PD2 derives a few inference rules based on the data dependencies between different software components, as well as between different nodes, and uses them to trace back each problematic flow. When tracing back the data paths, PD2 turns on the performance monitoring of, and collects debugging information from, only those modules and nodes that the problematic flows go through. Finally, PD2 visualizes the debugging information to locate the dominant sources (i.e., some software components on certain nodes) of data loss and latency on the data flows. PD2 does not intend to solve the performance problems, but helps users identify or further investigate the performance problems by locating them efficiently and accurately.

We have implemented PD2 on TinyOS and evaluated it on a real WSN testbed. Our experimental results show that PD2 can provide useful information to locate performance degradation caused by code bugs, weak links, and radio interference. Its energy cost is shown to be only 5–10% of that of collecting debugging information from all nodes.

CHAPTER 4

Distributed Location Service Protocol for Networked Stationary Sensors and Mobile Actors

4.1 Introduction

Many sensor network applications, such as habitat monitoring [1], emergency rescue, battlefield surveillance, border monitoring [66, 67], require interaction between stationary sensor nodes and mobile actors — a large number of resource-limited sensor nodes are deployed in a certain geographical area for physical-environment monitoring, and some mobile actors may move within the terrain and receive notification of events from the sensors. For example, in the emergency-rescue or military applications, emergency rescuers/vehicles or soldiers/military vehicles, as mobile actors, need to rescue missing people or track enemies. While moving inside the field, they interact with the sensor nodes to retrieve the location information of missing people or enemies.

Although mobiles often have a longer radio communication range than that of sensors, they may or may not form a connected graph/network, depending on their density and movement. Therefore, sensor networks should provide a routing mechanism to forward the sensing data to the mobiles, and the routing mechanism should perform well for a wide range of the mobiles' speeds. For example, foot-emergency-rescuers or foot-soldiers may move at low speed while ambulances or fire trucks, or military vehicles may move at high speed.

4.1.1 Background

There are two types of approaches to routing sensed data to a mobile actor (a.k.a. mobile sink): (1) TTDD [56] and Bread Crumb Routing [68] that do not require the knowledge of the

mobile's whereabout, and (2) GPSR [17, 18], Landmark Routing [19], and Beacon Vector Routing (BVR) [20] that require the knowledge of the mobile's location.

In (1), TTDD allows data sources (i.e., sensors) to proactively build grid structures over the entire network to disseminate the sensed data, so the overhead of publishing data may be amortized when there are many mobile sinks. Bread Crumb Routing assumes a mobility model under the constraint that the sensors marked by the mobile sink must form a connected path. However, in the above-mentioned scenarios, this path may be disconnected because (i) the mobile fails to leave marks on sensors as a result of message loss; (ii) the path may run through a deployment *hole*, and thus, the sensor marked by the mobile cannot communicate directly with any of previously-marked sensors; (iii) a sensor node on the path may fail or be destroyed.

In (2), the mobile needs to periodically report its geographic location or virtual address to selected nodes, called *location servers*, in order to use GPSR, Landmark Routing, or Beacon Location Service (BLS) [69]. Other nodes can acquire the mobile's location from one of its location servers and then deliver data to the mobile sink using one of these routing methods.

A number of location-service protocols have been proposed for mobile ad hoc networks (MANETs). GLS [21], DLM [22], HLS [70], and MLS [71] are hierarchical location service protocols, i.e., the mobile sink constructs a hierarchy of location servers over a grid structure. XYLS [72] lets the mobile sink select a *thick column* of nodes as its location servers. Twins [73], Home-Zone-Based Location Service [74], and GHLS [23] all use hash functions to select a centralized location server. Particularly, GHLS hashes the ID of the mobile sink into a geographic location, and the node closest to that location serves as the central location server for the mobile. Compared to XYLS and GLS, GHLS has the advantage of simplicity and very low overhead, and outperforms GLS for networks of up to 25,000 nodes, even though GLS asymptotically scales better [23].

These location service protocols, however, are not applicable to sensor networks due to the usually high per-hop latency in a sensor network which ranges from a few hundred milliseconds to a few seconds [75, 76], while that of a MANET is an order-of-magnitude lower (tens of *ms*) [77, 78]. The high per-hop latency in a sensor network can be attributed to scheduling delay and transmission time. First, wireless communication consumes much more energy than other operations for (severely energy-constrained) sensor nodes. Hence, energy-efficient MAC protocols avoid idle listening and overhearing by scheduling transmission and listening periods (e.g., S-

MAC [79] and T-MAC [80]), or low-power channel polling (e.g., WiseMAC [81] and BMAC [34]), or both (e.g., SCP [75]). As a result, the radio's duty cycle can be limited to a few percentages. Thus, a packet has to be held for a certain period of time before taking its next hop. Second, a sensor node's radio usually has a lower bandwidth, incurring a longer transmission time. For example, Mica2 (MicaZ) has a bandwidth of 19.5 kbps (250 kbps), while MANETs typically use wireless LAN cards of 11 Mbps or 54 Mbps.

This high per-hop latency makes packet transmission in a sensor network much slower than in a MANET. Moreover, a sensor network is usually of much larger scale than a MANET. Therefore, the location-service protocols intended for MANETs are unlikely to perform well in sensor networks, because, while a message is being delivered from its source to a location server, then to the mobile receiver's location obtained from the location server, the mobile could have moved too far away to receive the message directly as in GHLS or even by using forward pointers as in GLS. This problem becomes more evident as mobile moves faster.

4.1.2 Proposed Approach

In this chapter we present a *distributed location service protocol* (DLSP) for a hybrid wireless network of stationary sensors and mobile actors. Like GLS, DLSP is built on a hierarchical grid structure. A mobile selects multiple location servers at each level of the hierarchy, and sends location updates more frequently to the lower-level location servers than the higher-level ones. A location query (that also contains a data packet to be delivered) may take multiple rounds of "lookup-and-chase" to reach the mobile receiver. Through a rigorous analysis, we derive the condition under which a high query-delivery ratio (i.e., the data-delivery success rate in DLSP) is achieved, and show how to configure the protocol parameters to ensure the scalability of the location service. Here 'scalability' means that, as the network size increases, the location service protocol preserves the high query-delivery ratio and the protocol overhead is proportional to $O(log(N))$, where N is the network size. We find that, in order to preserve a high query-delivery ratio, the mobile's speed should be below a certain fraction of the packet-transmission speed, which depends on the underlying movement threshold. For example, if the movement threshold for the lowest-level location servers is the same as the node's radio range, the mobile's speed limit is a one-tenth of the packet-transmission speed. The theoretical speed limit is a one-fifth of the packet-transmission

speed beyond which DLSP does not scale regardless of the movement threshold.

Like GLS, DLSP incurs a high location-update overhead because a mobile needs to update multiple location servers at each level with its location information. To alleviate this problem, we propose an optimization, called *DLSP with a Selected Neighbor* (DLSP-SN), in which the mobile updates the location server in at most one neighbor square at each level. A neighbor square is selected based on the mobile's trajectory. DLSP-SN achieves a significant reduction of update overhead. However, due to the gridding effect,[1] DLSP-SN may incur more rounds of lookup-and-chase than DLSP, thus making the average path length of location queries greater than that of DLSP and increasing data-delivery cost. In order to make a tradeoff between location-update and data-delivery costs, we present a greedy adaptation mechanism, called DLSP-ASN, to improve the overall energy-efficiency.

The contributions of this chapter are summarized as follows.

- Design of DLSP: We design a novel hierarchical location service. In DLSP, location-updates are published to hierarchical location servers, and location-queries are processed recursively using these hierarchical location servers. DLSP can efficiently provide mobiles' location information with wide range of mobiles' speeds even in the presence of sensor node failures.

- Optimization of DLSP: We provide two optimized algorithms for DLSP, DLSP-SN and DLSP-ASN. The former focuses on reducing the location-update overhead, while the latter makes a good balance between the location-update overhead and the data-delivery ratio.

- Evaluation of DLSP: We rigorously and throughly evaluate DLSP and its optimizations. First, we derive the condition under which DLSP guarantees a high data-delivery ratio using a mathematical analysis. Second, we extensively simulate DLSP with various scenarios and parameters to show its performance in diverse environments.

The rest of this chapter is organized as follows. Section 4.2 describes the details of DLSP. Section 4.3 derives the condition for DLSP to achieve a high packet-delivery ratio, while Section 4.4 analyzes the overhead of DLSP, and presents an enhanced version of DLSP, called DLSP-SN.

[1] 'Gridding effect' means that the source and destination nodes across but close to the boundary of a high-level square may require the query to travel many hops upward (in the hierarchy) to the common (parent) square containing both nodes. Both GLS and DLSP-SN suffer from the gridding effect, but DLSP does not.

Section 4.5 proposes a greedy adaptation mechanism, DLSP-ASN. In Section 4.6 we use simulation to evaluate the performance of location services. We conclude the chapter and discuss future directions in Section 4.7.

4.2 Distributed Location Service Protocol

We now present the details of DLSP. We assume that a large number of stationary sensors have been placed randomly and uniformly in a field of interest and a relatively smaller number of mobile actors move around within the field. Geographic routing (e.g., GPSR [17]) is used for multi-hop routing. Each sensor node can determine its location by using a GPS receiver or a localization service [82, 83]. Likewise, each mobile either is equipped with a GPS receiver or can estimate its location using the neighbor sensors' location information.

Table 4.1 lists the notation used in this chapter.

4.2.1 Selection and Update of Location Servers

A sensor network is assumed to have been deployed in a square field as in GHT [84]. Similar to GLS [21], the entire square field is partitioned into a grid as shown in Figure 4.1. Four level-0 squares make up one level-1 square, four level-1 squares make up one level-2 square, and so on. To avoid overlap between two squares of the same size, a particular level-k square is part of one and only one level-$(k+1)$ square. For simplicity, we assume that the field is perfectly gridded, i.e., the field is a square of edge length $\mathrm{L} = 2^h \ell$. We will discuss how this restriction can be relaxed in Section 4.6. Each node is pre-loaded with h, ℓ, and the location of the lower-left corner of the field, and it can calculate the entire grid structure using this information.

Suppose a mobile R needs to send its location updates at time T (we will later elaborate on when to send location updates). It selects a location server in its level-0 square and also the neighbor squares, denoted as $LS_{0,j}(R,T)$ $(j = 0,\ldots,8)$. To randomize the selection, R uses a common hash function[2] to compute a location in a square, and the sensor node closest to that location is chosen as the R's location server. A neighbor square is omitted if it is outside of the field boundary. At level-1, R picks a location server from each of the neighbor squares, $S_{1,j}(R,T)$. There is no location server in $S_{1,0}(R,T)$, as it is fully covered by the level-0 location servers, and so on.

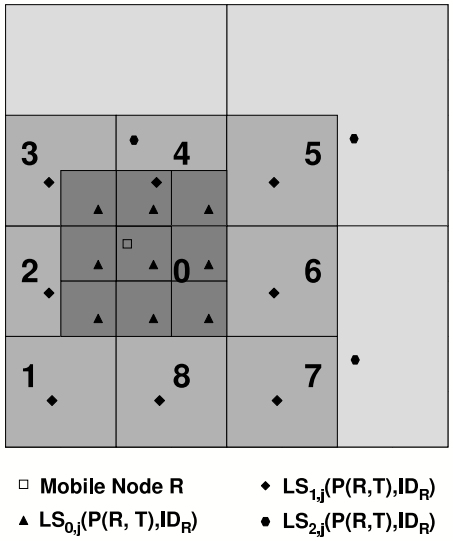


Figure 4.1: The location servers selected at three levels of the grid

Location servers at *different* levels are updated at *different* rates. Suppose at time T, R has sent a location update (i.e., $P(R,T)$) to level-k location servers. It will then send the next update to the level-k servers at $T+\Delta T$ if and only if $dist(P(R,T),P(R,T+\Delta T)) \geq 2^{k-m}\ell$ (i.e., the movement threshold) or $\Delta T \geq 2^k\tau$ (i.e., the timeout). R sets the *lifetime* of its location servers to be slightly larger than $\Delta T = min(2^k\tau, \frac{2^{k-m}\ell}{\bar{v}})$. If a location server does not receive a new update from the mobile R before this lifetime expires, it is no longer a location server for R.

The hash function lets different mobile sinks choose different sensor nodes as their location servers. As a result, the protocol evenly distributes the workload and energy consumption among the sensor nodes.

4.2.2 Processing of Location Queries

When a sensor node S sends a data message to R, it only knows R's ID. First, it tries to find R's location from its neighbor's table and local location cache (i.e., it is a location server for R). If R's location is not found, S encapsulates the data into a location query, and sends it to a location

[2]The hash function can be defined in many different forms. For example, $H(R,Sq) = (f_x(R)\cdot 2^k\ell, f_y(R)\cdot 2^k\ell)$, where f_x and f_y are uniform distribution functions within the interval $(0,1)$, and Sq is a level-k square. Note that the hash function computes a relative location inside Sq.

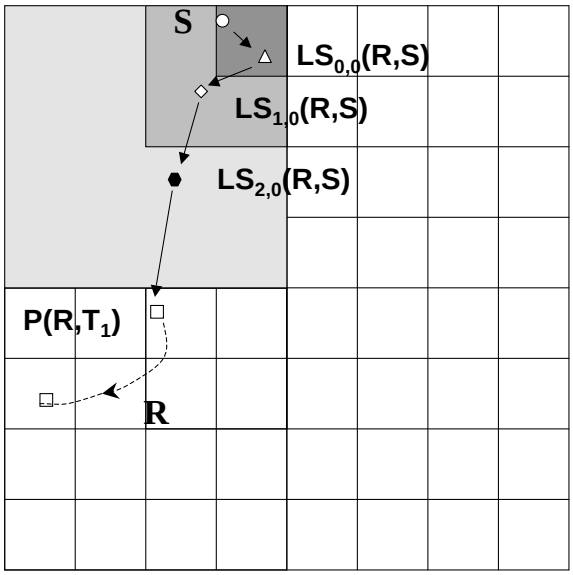


Figure 4.2: Round 1 of location query processing

server. Once R's location is found, the data message is sent to that location using geographic-location-based-routing. This *lookup-and-chase* process is illustrated by an example in Figures 4.2 and 4.3.

In Figure 4.2, S first assumes that R has visited somewhere nearby — R and S are in the same level-0 square or two adjacent level-0 squares. Formally, S assumes $L_{0,j}(R,T)$ to be $L_{0,0}(S)$. So, the location query is sent to $LS_{0,0}(R,S)$. However, $LS_{0,0}(R,S)$ does not have R's location information, so it tries to find R's location in a larger square by sending the query to $LS_{1,0}(R,S)$, and so on. Eventually, $LS_{2,0}(R,S)$ has R's location information at time T_1 (i.e. $LS_{2,0}(R,S)$ is also $LS_{2,4}(R,T_1)$) , denoted as $P(R,T_1)$, so it sends the query to $P(R,T_1)$. This process of looking for the location of, and chasing, the mobile is called a *round*.

If R moves fast and if S and R are far apart, by the time the location query reaches the location $P(R,T_1)$, R could have moved too far away from $P(R,T_1)$ to receive the location query. In such a case, the query will be received by the node A closest to $P(R,T_1)$. Unlike GLS, A does not maintain any forward pointer under DLSP. Instead, it starts a new *round*. As shown in Figure 4.3, the query first goes to $LS_{0,0}(R,A)$, then to $LS_{1,0}(R,A)$ (i.e., $LS_{1,6}(R,T_2)$), which has more recent R's location information, $P(R,T_2)$. Finally, the query catches up with R near $P(R,T_2)$.

After receiving the query, R may decide whether or not to send its location information to

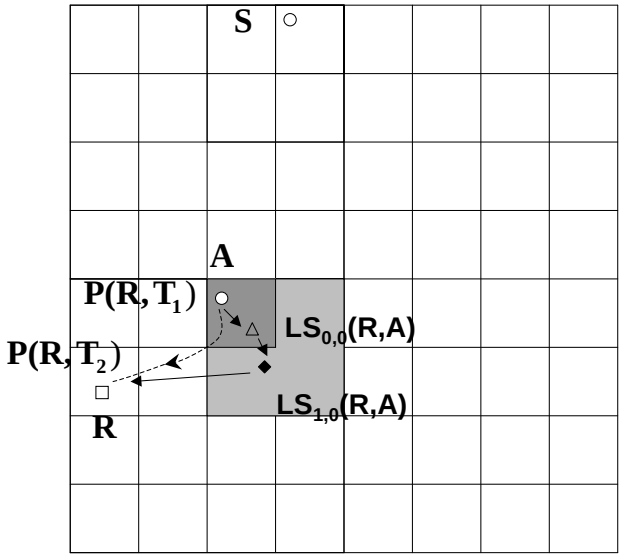


Figure 4.3: In round 2, only the location server in the shaded level-1 neighbor square is visited

S, which caches the location for later queries. Such a decision should depend on the sender's transmission rate, as discussed in Section 4.5.

4.3 Conditions for High Packet-Delivery Ratio

In this section, we first derive the condition for achieving a high packet-delivery ratio under DLSP. Then, we discuss how to configure the parameters of DLSP to make it scalable. DLSP is found to be scalable if the mobile's speed is lower than a certain fraction of the packet-transmission speed, which depends on the movement threshold used. Finally, we present the condition for achieving a high packet-delivery ratio in GHLS, and also show that GHLS is not scalable.

4.3.1 Conditions for High Packet-Delivery Ratio under DLSP

Our analysis of DLSP consists of the *base* case and the *inductive* step. The base case analyzes how a location query can catch up with the mobile receiver after obtaining its location information from a level-0 location server. The inductive step analyzes how the location query can get closer to the mobile by completing each round.

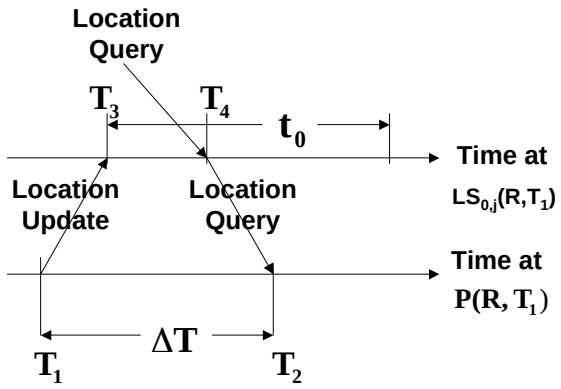


Figure 4.4: The timeline of events for location query processing at level-0.

The Base Case

Suppose, at time T_1, R sends its location, $P(R,T_1)$, to a level-0 location server, $LS_{0,j}(R,T_1)$, $j \in \{0,1,\ldots,8\}$. The location server receives the location update at time T_3. At time T_4, it receives a location query and forwards the query to $P(R,T_1)$. The location query reaches location $P(R,T_1)$ at time T_2. The timeline of these events are shown in Figure 4.4.

In order to have R receive the query at T_2, the following condition must be satisfied:

$$dist(P(R,T_1),P(R,T_2)) \leq r. \tag{4.1}$$

Suppose $\Delta T = T_2 - T_1$, then $dist(P(R,T_1),P(R,T_2))$ is bounded by $\Delta T\bar{v}$, because the distance is maximized when R moves on a straight line between T_1 and T_2. The average speed is computed as the length of the trajectory curve between T_1 and T_2 over ΔT. ΔT can be broken into three components, $T_3 - T_1$, $T_4 - T_3$, and $T_2 - T_4$. $T_3 - T_1$ denotes the average latency of the location update from $P(R,T_1)$ to $LS_{0,j}(R,T_1)$; $T_4 - T_3$ represents the average obsoleteness of the location information at the location server; $T_2 - T_4$ denotes the average latency of the location query from $LS_{0,j}(R,T_1)$ to $P(R,T_1)$.

Let d_0 be the average distance between R and a level-0 location server, i.e., $dist(P(R,T_1),L_{0,j}(R,T_1)$. Then, considering R as a random point in an $\ell \times \ell$ square, and the location server as a random point in the same square or one of the eight adjacent squares, we get $d_0 \approx 1.27\ell$ according to a numerical analysis. Also, we let t_0 be the update interval for level-0 location servers. We have

$T_3 - T_1 = T_2 - T_4 = \frac{d_0}{p} t_h$, and $T_4 - T_3 = \frac{1}{2} t_0$ because T_4 ranges from T_3 to $T_3 + t_0$. So,

$$\Delta T = \frac{1}{2} t_0 + 2\frac{d_0}{p} t_h. \tag{4.2}$$

Also, from Section 4.2, we have

$$t_0 = \begin{cases} \tau & \text{if } \bar{v} < \frac{2^{-m}\ell}{\tau} \\ \frac{2^{-m}\ell}{\bar{v}} & \text{if } \bar{v} \geq \frac{2^{-m}\ell}{\tau}. \end{cases} \tag{4.3}$$

From Eq. (4.3), we have

$$\bar{v} t_0 \leq 2^{-m}\ell. \tag{4.4}$$

Therefore,

$$dist(P(R,T_1), P(R,T_2)) \leq \frac{1}{2} t_0 \bar{v} + 2\frac{d_0}{p} t_h \bar{v} \tag{4.5}$$

In order to satisfy Eq. (4.1), we simply let $\frac{1}{2} t_0 \bar{v} + 2\frac{d_0}{p} t_h \bar{v} \leq r$. That is,

$$\begin{cases} \tau\bar{v} + \frac{5.08\ell}{p} t_h \bar{v} \leq 2r & \text{if } \bar{v} < \frac{2^{-m}\ell}{\tau} \\ 2^{-m}\ell + \frac{5.08\ell}{p} t_h \bar{v} \leq 2r & \text{if } \bar{v} \geq \frac{2^{-m}\ell}{\tau}. \end{cases} \tag{4.6}$$

Approximately, Eq. (4.6) can be satisfied if

$$2^{-m}\ell + \frac{5\ell}{p} t_h \bar{v} \leq 2r. \tag{4.7}$$

Analysis of the Inductive Step

Consider the case of requiring multiple rounds of lookup-and-chase. Suppose the query looks up R's location from a level-k_i location server in round i, and from a level-k_{i+1} server in round $i+1$. To ensure the query makes progress toward R, we need to satisfy

$$k_{i+1} \leq k_i - 1. \tag{4.8}$$

Suppose the query returns R's location information, $P(R,T_1')$ in round i and reaches $P(R,T_1')$ at time T_2'. $k_{i+1} \leq k_i - 1$ holds if the following inequality holds:

$$dist(P(R,T_1'),P(R,T_2')) \leq 2^{k_i-1}\ell. \tag{4.9}$$

Eq. (4.9) bounds the distance between the known location of round i and that of round $i+1$, so the two locations are at most in two adjacent level-(k_i-1) squares. Therefore, the location server visited at round $i+1$ is at most of level-(k_i-1).

Similar to Eq. (4.2), we get

$$\Delta T' = T_2' - T_1' = \frac{1}{2}2^{k_i}t_0 + 2\frac{2^{k_i}d_0}{p}t_h. \tag{4.10}$$

So, we have

$$dist(P(R,T_1'),P(R,T_1')) \leq \frac{1}{2}2^{k_i}t_0\bar{v} + 2\frac{2^{k_i}d_0}{p}t_h\bar{v} \tag{4.11}$$

In order to satisfy Eq. (4.8), we simply let $\frac{1}{2}2^{k_i}t_0\bar{v} + 2\frac{2^{k_i}d_0}{p}t_h\bar{v} \leq 2^{k_i-1}\ell$. That is,

$$\begin{cases} \tau\bar{v} + \frac{5.08\ell}{p}t_h\bar{v} \leq \ell & \text{if } \bar{v} < \frac{2^{-m}\ell}{\tau} \\ 2^{-m}\ell + \frac{5.08\ell}{p}t_h\bar{v} \leq \ell & \text{if } \bar{v} \geq \frac{2^{-m}\ell}{\tau}. \end{cases} \tag{4.12}$$

Again, due to Eq. (4.4), Eq. (4.12) can be satisfied if

$$2^{-m}\ell + \frac{5\ell}{p}t_h\bar{v} \leq \ell. \tag{4.13}$$

4.3.2 Configuration of Protocol Parameters for DLSP

The above analysis provides insights into which parameters affect the packet-delivery ratio and how they can be configured to achieve the scalability of DLSP with respect to query delivery.

Configuration of ℓ

Consider the condition of the base case, Eq. (4.7), and that of the inductive step, Eq. (4.13). The condition of the base case is stronger than that of the inductive step if $\ell \geq 2r$. Moreover, both

Eqs. (4.7) and (4.13) are independent of the field edge length, L. Therefore, as long as data can be delivered within a small region (level-0 squares) of edge length $\ell \geq 2r$, it can be delivered from an arbitrarily far away node. In fact, we need

$$\ell = 2r \tag{4.14}$$

because the overhead of location updates increases as ℓ increases (in Section 4.4).

Configuration of m

In Eq. (4.7), $\frac{5\ell}{p} t_h v$ is always positive since t_h is not negligible. So, m must be a positive integer. Again, the overhead of location updates is proportional to 2^m when the mobile's speed is above the threshold. Therefore, m should be set to 1, and the movement threshold is r.

Mobile's Speed Limit

From Eq. (4.7), if $m = 1$, $\bar{v} < \frac{r}{5\ell} \frac{p}{t_h} = \frac{p}{10t_h}$, which is a one-tenth of the packet-transmission speed. If the movement threshold for location updates gets smaller, the location updates become more frequent, and the mobile is allowed to move faster. However, $\bar{v} < \frac{2r}{5\ell} \frac{p}{t_h}$ must always hold, and the speed can never be greater than $\frac{p}{5t_h}$. So, the mobile's theoretic speed limit is a one-fifth of the packet transmission speed, no matter how frequently the location servers are updated.

4.3.3 Choice of Design Paradigm

GHLS (i.e., a centralized paradigm) can be considered as a trivial case of DLSP (i.e., a hierarchical paradigm), in which $\ell = \mathrm{L}$. The analysis of GHLS is the same as that of the base case in DLSP, except that $d_0 \approx 0.5\mathrm{L}$ because the mobile and its location server are considered two random points in the $\mathrm{L} \times \mathrm{L}$ square.

Suppose the movement threshold for updating the location server is $\bar{v}t_0 \leq d$. We need to satisfy

$$d + \frac{2\mathrm{L}}{p} t_h \bar{v} \leq 2r. \tag{4.15}$$

When t_h is not very small, Eq. (4.15) may not hold for large networks and fast mobiles.

Based on Eqs. (4.15), (4.4), and (4.14), we can conclude that, regardless of the mobile's speed,

the conditions of GHLS (i.e., the centralized paradigm) are stronger than those of the hierarchical paradigm if $L < 2.5l$. So, the centralized paradigm is favorable for low mobile's speed, very low per-hop packet latency, or small/ median networks because of its simplicity and lower overhead [23]. For large networks with high mobility and non-trivial packet latency, the hierarchical paradigm should be used.

4.4 Analysis of Location-Service Overhead

In this section, we first analyze the overhead of location updates under DLSP and then propose a design optimization, called DLSP with a Selected Neighbor (DLSP-SN), which significantly reduces the location-update overhead.

4.4.1 Analysis of Location-Update Overhead

Let U denote the total overhead of location updates, and u_k the overhead of updating a level-k location server. The location-update frequency for level-k location servers is $t_k = 2^k t_0$. The average distance between R and a level-k location server ($LS_{k,j}(R,T)$ is $1.27 \cdot 2^k \ell$, and that between R and the level-0 location server $LS_{0,0}(R,T)$ is 0.5ℓ. Since there are at least 3 neighbor squares at each level except the highest, we have

$$\begin{aligned} U &\geq \sum_{k=0}^{h-1} 3 \cdot 1.27 \cdot 2^k \ell \frac{1}{t_k} + 0.5\ell \frac{1}{t_0} \\ &\geq (3.8h + 0.5) \frac{2^m \bar{v}}{p} \end{aligned} \tag{4.16}$$

where $h = O(log(\mathrm{L} \times \mathrm{L}))$, and the total number of nodes, N, is proportional to $\mathrm{L} \times \mathrm{L}$ for a given node density. So, $U = O(log(N))$. That is, DLSP is asymptotically scalable w.r.t. the protocol overhead. However, like GLS, DLSP suffers from high update overhead because there are multiple location servers at each level of the hierarchy.

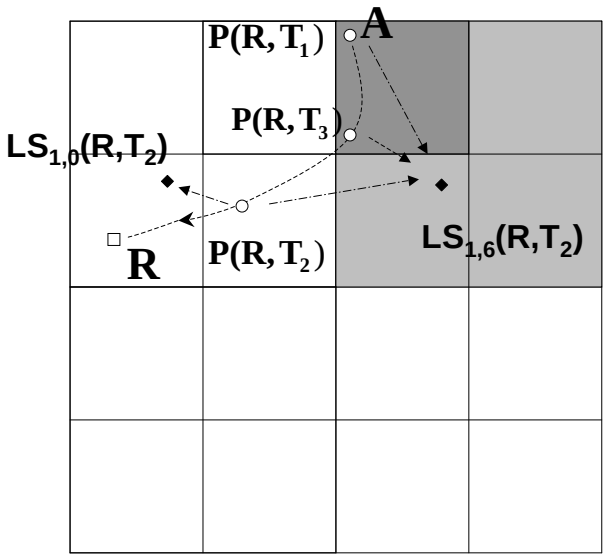


Figure 4.5: R sends updates to two level-1 location servers at $P(R,T_2)$, because $P(R,T_3)$ is in the selected neighbor square of $P(R,T_2)$.

4.4.2 Optimization of DLSP

Our optimization goal is to reduce the location-update overhead while preserving the high packet-delivery ratio. The key observation is that it is unnecessary to update the location servers in all neighbor squares. This is because, as a location query "chases" the mobile receiver, the mobile's trajectory determines which location servers to communicate with.

This observation is illustrated in Figure 4.3. At time T_2, R updates the five location servers in the neighbor squares. Therefore, at round 2, the query can obtain a more recent location, $P(R,T_2)$, and catch up with R. Since A is in $S_{1,6}(P(R,T_2))$, the query relayed by A can only go through $LS_{1,6}(R,T_2)$, not the other four level-1 location servers. That is, only the update to the location server in the neighbor square, $S_{1,6}(R,T_2)$ is useful for delivering this query. So, the design optimization is called *Distributed Location Service Protocol with a Selected Neighbor* (DLSP-SN).

To illustrate how DLSP-SN works, let us zoom in the lower-left level-2 square of Figure 4.3 in Figure 4.5. Suppose R needs to send location updates to level-1 location servers at $P(R,T_1)$, $P(R,T_3)$, and $P(R,T_2)$ consecutively. At $P(R,T_3)$, it checks if its previous location $P(R,T_1)$ was in the level-1 square, $S_{1,0}(P(R,T_1))$. If so, it only updates $LS_{1,0}(R,T_1)$ (i.e., $LS_{1,6}(R,T_2)$. At $P(R,T_2)$, R finds that its previous location $P(R,T_3)$ is in the neighbor square, $S_{1,6}(P(R,T_2))$, so it sends updates to both $LS_{1,0}(R,T_2)$ and $LS_{1,6}(R,T_2)$. Note that the locations of two consecutive level-k

updates must be in the same level-k square or two neighbor level-k squares, because the movement threshold for level-k updates, $2^{k-m}\ell$, is strictly less than the edge length of level-k square, $2^k\ell$.

The difference between DLSP and DLSP-SN is summarized as follows. First, suppose the highest level is h. Then, DLSP updates $LS_{0,j_1}(R,T)$ ($j_1 = 0,1,\ldots,8$), and $LS_{k,j_2}(R,T)$ ($k = 1,2,\ldots,h-1$ and $j_2 = 0,1,\ldots,8$). DLSP-SN updates $LS_{k,0}(R,T)$ ($k = 0,2,\ldots,h$), as well as the location server in the selected neighbor square. Second, suppose k_i and k_{i+1} are the levels of location servers DLSP and DLSP-SN obtains location information at rounds i and $i+1$. DLSP requires $k_i > k_{i+1}$, but DLSP-SN does not have this restriction. To avoid endless chasing, DLSP-SN requires that, at each round, the query get more recent location information than the previous round.

DLSP-SN is less restrictive in the sense of obtaining location information, because it selects much fewer location servers than DLSP. As a result, DLSP-SN incurs more rounds and longer query paths.

4.5 Adaptation of Location Service

DLSP-SN reduces its update overhead, but may extend the query path length, increasing the data-delivery cost. This increase of data-delivery cost may become significant in case of continuous data streams commonly seen in sensor network applications. To achieve overall energy-efficiency with DLSP-SN, we propose an adaptive location-update scheme in which a mobile adaptively sends its location updates based on the varying distribution and rate of the data sources. We then analyze the parameter configuration for the adaptation to ensure a high query-delivery ratio and present a greedy algorithm to improve overall energy-efficiency. Finally, we summarize the comparison among the hierarchical location service protocols, DLSP, DLSP-SN, DLSP-ASN, and GLS.

4.5.1 Adaptive Location Updates

In MANETs, most data communications are one-to-one. After the mobile receives a location query, it can periodically send location updates directly to the source node. The source can cache the location information and send data directly to the mobile until the location information expires.

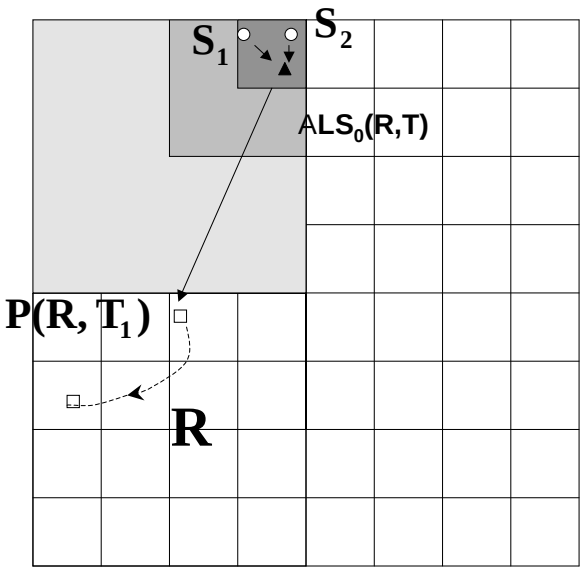


Figure 4.6: Location queries (or data packets) from S_1 and S_2 travel less hops during *round* 1 with adaptive location updates.

In a hybrid wireless network of stationary sensors and mobile actors, however, a mobile may receive data from multiple data sources located in the areas of interest. If the mobile receiver has to send location updates to each of these data sources, the location-update overhead can be prohibitive. Fortunately, the data sources in an area of interest may be spatially close to one another. Therefore, the mobile can send updates to only a few location servers shared by the data sources.

Figure 4.6 provides an illustrative example. Suppose sensor nodes, S_1 and S_2, reside in the same level-0 square $S_{0,0}(P(S_1))$, and continuously report data to a mobile R. Instead of sending location updates to S_1 and S_2 individually, R picks an adaptive location server, $ALS_0(P(R,S_1)$, in $S_{k,0}(P(S_1))$, and periodically sends updates to it as well as to the other location servers. When S_1 or S_2 sends R data, a location query is processed exactly the same as in Section 4.2 except for the first round. R's location can be obtained from $ALS_0(R,S_1)$, instead of from the level-2 location server, $LS_{2,j}(R,T_1)$, as shown in Figure 4.2. Thus, the data-delivery cost is reduced at the expense of extra adaptive location updates.

4.5.2 Condition for High Query-Delivery Ratio

Suppose R updates $ALS_0(P(S_1), ID_R)$ at time T_1', and the location query reaches $P(R, T_1')$ at time T_2'. Let $T_a = 2^{k_a} t_0$ be the period of location updates to $ALS_0(R, S_1)$, and D be the distance between $L(R, S_1)$ and $P(R, T_1')$.

Similar to the analysis of Eq. (4.10), we get

$$\Delta T' = T_2' - T_1' = \frac{1}{2} 2^{k_a} t_0 + 2\frac{D}{p} t_h. \tag{4.17}$$

We simply let $\Delta T' \bar{v} \leq \frac{D}{2}$. That is,

$$2^{k_a - 1} t_0 \bar{v} + 2\frac{D}{p} t_h \bar{v} \leq \frac{D}{2}. \tag{4.18}$$

In Section 4.3, we derived Eq. (4.4) and the speed limit, $v < \frac{p}{10 t_h}$ with the movement threshold r. In order to satisfy Eq. (4.18), we need

$$2^{k_a - 2} \ell + \frac{D}{5} \leq \frac{D}{2}. \tag{4.19}$$

$$k_a \leq log_2(\frac{D}{\ell}). \tag{4.20}$$

When k_a is small, there is higher update overhead but lower data-delivery cost; when k_a is large, there is less update overhead but higher data-delivery cost. So, k_a needs to be configured to achieve overall energy-efficiency.

4.5.3 Analysis of Overall Energy-Efficiency

Let E denote the energy-efficiency without adaptive location updates, and E_a denote the efficiency with adaptive location updates. The mobile maintains a moving window to compute the average data rate, R_{data}, the average hop count, C_{hops}, and the average distance, D_{src}, from the two sources.

$$\Delta E = R_{data} \cdot \left(C_{hops} - \frac{D_{src}}{p} \right) - \frac{D_{src}}{p} \frac{1}{2^{k_a} t_0} \tag{4.21}$$

A mobile periodically searches for k_a in $[0,\ log_2\left(\frac{D}{\ell}\right)]$ such that ΔE is maximized, or E_a is minimized.

In general, the rate and distribution of data sources cannot be described in a simple form. It is, therefore, very difficult to compute the optimal solution that combines data sources and sets k_a for each combination. The mobile can use some simple heuristics to find good solutions. For example, only the data sources in the same level-0 are combined at their level-0 location server. We can then use the above analysis to set k_a for each level-0 location server.

4.5.4 Comparison of Hierarchical Location Services

Different hierarchical location services are compared and summarized in Table 4.2.

4.6 Evaluation

We have performed extensive simulation to comparatively evaluate the performance of location-service protocols. For this purpose, we have implemented the DLSP protocols (DLSP, DLSP-SN) and GHLS in *ns-2* [85], and also have ported GLS to the same version of *ns-2* we used for other protocols.

The following metrics are evaluated for the location service protocols: (1) Query Delivery Ratio—the percentage of location queries successfully delivered to the mobile receiver; (2) Update Overhead—the number of update packets transmitted with each hop counted as one packet transmission; (3) Query Path Length—the number of hops each successfully-delivered query takes.

[3] In GLS, one square considers its three adjacent squares belonging to the same parent square as its neighbors; in DLSP protocols, all the eight adjacent squares are considered neighbors.

[4] 'Gridding effect' means that the source and destination nodes across, but close to, the boundary of a high-level square may require the query to travel multiple hops upward in the hierarchy to the common parent square that contains both nodes.

[5] Restrictions: (1) The query needs to obtain a newer location of the mobile at each round; (2) the query does not proceed if the location is obtained from a level-0 location server in the previous round; (5) the level of the location servers gets lower as the round progresses.

4.6.1 The Simulation Setup

The transmission range for radio communication is set to 100m based on the characteristics of MicaZ [86] devices. We assume the radio link is symmetric, and only collision may cause message loss. Typically, the raw radio of sensor nodes (e.g., Mica2, MicaZ) is lossy and asymmetric, but the underlying MAC or routing protocols provide reliable transmission via scheduling and retransmission. We use both 802.11 MAC and S-MAC in our simulation.

Sensors are uniformly deployed over a square area, with density of 6.25 nodes per $100 \times 100\text{m}^2$. This high node density is chosen because in low node-density networks, geographic routing (e.g., GPSR) suffers from relatively high packet losses, which may distract the readers from our main focus on the performance of location services. Given this high node density, the average per-hop progress is approximately $0.7r$ or 70m. Our tests are run on networks of 400×400, 800×800, 1200×1200, and $1600 \times 1600\text{m}^2$, which include 100, 400, 900, and 1600 sensor nodes, respectively. Since interactions among mobiles are not considered, only one mobile is simulated in our evaluation, and its movement follows the modified random way-point mobility model [87] in our base-case scenarios. The mobile's speed ranges from 4 to 40m/s, and its pause time is set to 0. We also simulate DLSP with random walk and Guass-Markov mobility models [88] to confirm that the proposed protocol functions regardless of the mobiles' movement pattern. We also simulated DLSP using the scenarios with 100×100 and 200×200 m^2 *void* areas to show its resilience to node failures.

The beacon period is set to 10s for stationary sensor nodes and 1s for the mobile. When a sensor node receives a beacon from the mobile, it replies with a beacon after a random delay ranging from 0 to 1s. The movement threshold for triggering location updates in DLSP, DLSP-SN, GHLS, and GLS is set to 100m (i.e., $m = 1$). The timeout for triggering location updates for the location service protocols except for GLS (i.e., τ) is 8s. GLS does not have any timeout. Instead of using its instantaneous speed, the mobile uses its average speed over a moving window. Suppose R sends two consecutive updates to its level-k location servers at time T and T'. The average speed $\bar{v} = \frac{dist(P(R,T),P(R,T'))}{T'-T}$, although R's trajectory may follow an arbitrary curve. To determine the timeout for the location information sent to a level-k location server, the mobile uses the average speed to predict the update interval $t_k = 2^k t_0$ with Eq. (4.3).

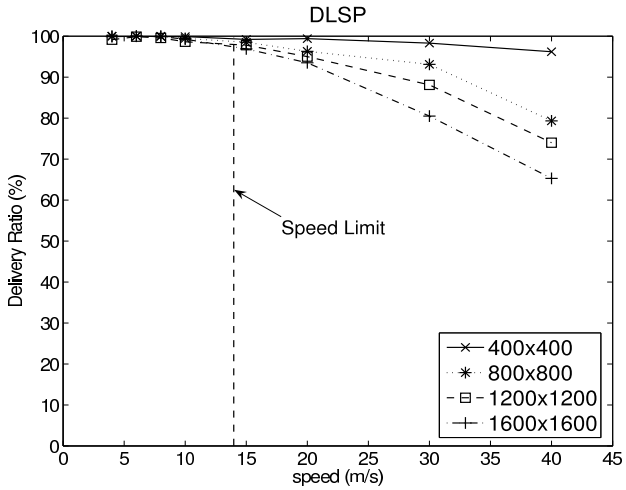


Figure 4.7: The query-delivery ratio of DLSP is higher than 96% for all network sizes if the mobile's speed $\leq 15m/s$. The speed limit from our analysis is 14m/s.

The edge length of the smallest square in the DLSP protocols (i.e., ℓ) is 200m. In GLS, the smallest square size is set to 100m, because all nodes in the same smallest square should be within two hops. The network size we tested, $1200 \times 1200\text{m}^2$, does not result in a perfect grid structure. In such a case, if an intended level-k square is not within the network boundary, it is substituted by a neighbor level-k square inside the boundary. For example, the level-2 square may be outside the boundary when the mobile is located at (900m,900m). Then, the level-2 square {(0,0), (800m,800m)} becomes its replacement.

Ten sensor-node deployments are generated for each network size. With each deployment, we generate a movement scenario for each speed. All test results are the averages of 10 runs on all the deployments. Since the mobile's ID is the same in all tests, a seed is randomly generated in each run so that a sensor node can hash the mobile's ID into a different value for DLSP and GHLS. As for the workload, a sensor node is randomly chosen to send a location query to the mobile once every 2s during a period of 200s, i.e., 100 queries are sent. All tests for the same network size use the same workload. In GLS, every node should publish its location information to its location servers for the correct functioning of GLS. For fair comparison, we modify GLS such that the sensor nodes publish their location only during the initial warm-up period of 120s. These location updates during the warm-up period are not counted in the update overhead.

For all protocols, the workload starts at 120s and the simulation ends at 300s. The surplus of

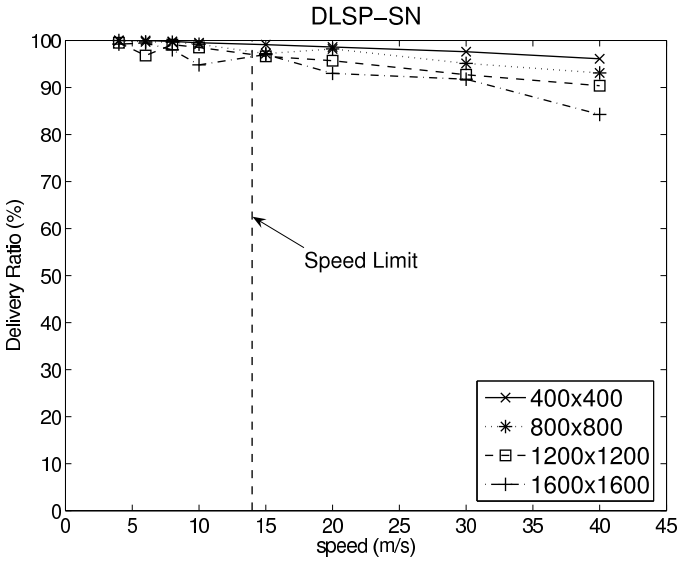


Figure 4.8: The query-delivery ratio of DLSP-SN is close to that of DLSP below the speed limit, and noticeably better in case of high speeds.

80s allows the last few queries to be delivered.

4.6.2 Simulation Results Using 802.11 MAC

Using the 802.11 MAC in *ns-2*, the transmission time plus the backoff delays ranges from 0.001 to 0.02s. Without a low-power MAC at hand, we add a fixed link-layer delay of 0.5s (or 0.25s). Thus, the actual per-hop latency ranges from 0.5 to 0.52s (or 0.25 to 0.27s), which resembles the per-hop latency in low-power MACs [75, 76]. Because the average per-hop progress of a packet is about 70m, the average packet-transmission speed is calculated to be 140m/s. Based on our analysis, the speed limit with the movement threshold of 100m is 14m/s.

Query-Delivery Ratio

Figure 4.7 shows that DLSP scales well if the mobile's speed $\leq$ 15m/s. In the network of 1600 nodes, the delivery ratios of both DLSP and DLSP-SN drop below 90% beyond the theoretic speed limit, 28m/s. We have also run tests with different per-hop latencies and different movement thresholds. The results are consistent with our analysis, and thus omitted.

The query-delivery ratio of DLSP-SN, as shown in Figure 4.8, is close to that of DLSP below

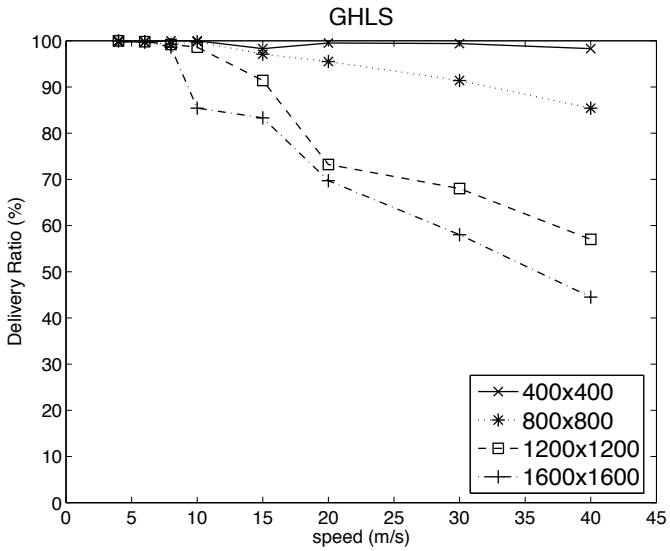


Figure 4.9: There is no single speed limit for different network sizes in GHLS because it does not scale.

20m/s and even higher than that speed because DLSP requires the query to obtain location information from a lower-level location server than the previous round, but DLSP-SN does not have this restriction and can take more lookup-and-chase rounds. Figure 4.9 shows that the delivery ratio of GHLS degrades significantly as the network size and the mobile's speed increase. This is because, as the per-hop latency is non-trivial, the term $\frac{2L}{p}t_h\bar{v}$ easily exceeds the bound, $2r$, in Eq. (4.15). When the query reaches the location obtained from the location server, the mobile has already moved too far away from that location to receive the query, and hence the message is dropped.

The delivery ratio of GLS, shown in Figure 4.10, degrades significantly as the network size and the mobile's speed increase, also because the mobile has moved too far away to receive the query when it reaches the location. In GLS, the mobile attempts to leave a forwarding pointer in the grid of which it moves out, so that a query may follow the mobile using the forwarding pointers. But the messages containing the forwarding pointers are likely to get lost, particularly when the mobile moves fast, because the destination of these messages (i.e., the grid it moves out of) is in the opposite direction of the node's movement. By geographic forwarding, the mobile selects the neighbor closest to the destination. But such a neighbor is most likely to be out of the mobile's radio range. When a forwarding pointer is lost, the chain of forwarding pointers is broken, and the query has to be dropped.

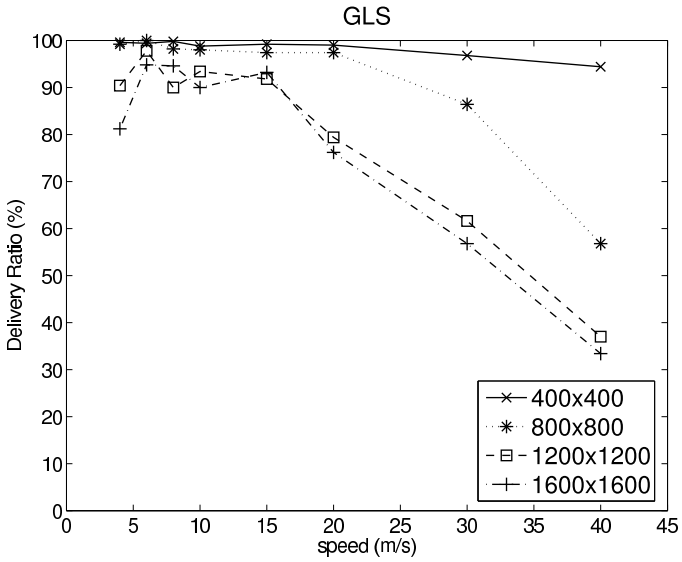


Figure 4.10: The delivery ratio of GLS degrades because many forward pointer messages are lost.

GLS also suffers some performance degradation at a low speed for the following reason. Unlike the other location protocols we evaluate, location updates are triggered only by the movement threshold in GLS. So, when the mobile's speed is low, the update period is very long, especially for high-level location servers in large networks. Then, loss of a location update can cripple these location servers for a very long time. Queries will be dropped if they reach these servers. At high speeds, the delivery ratio of small networks is noticeably better than that of large networks, because it is easier for a query to catch up with the mobile within smaller areas.

Location-Update Overhead

Because GHLS is shown in [23] to incur the least update overhead, we normalize the update overheads of DLSP, DLSP-SN, and GHLS with respect to that of GHLS, as illustrated in Figures 4.11–4.13. We obtained the results from the same tests for the query-delivery ratio. All the normalized overheads are relatively insensitive to the mobile's speed, since the tests of all protocols use the same movement threshold for triggering location updates. As the mobile's speed increases, the update overhead increases accordingly for all protocols.

Compared to DLSP, DLSP-SN reduces the update overhead by at least 70%, as shown in Figures 4.11 and 4.12. More importantly, the normalized overhead of DLSP-SN decreases as the net-

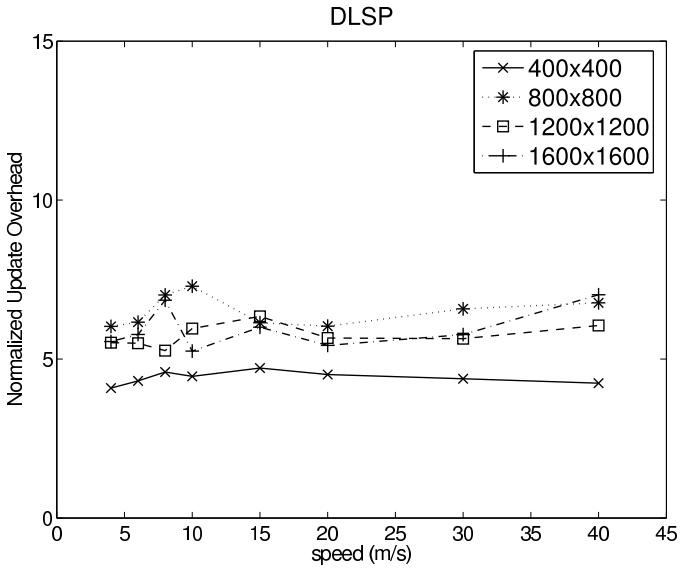


Figure 4.11: DLSP incurs a very high update overhead because there may be as many as 8 location servers at each level.

work size increases because the normalized overhead of DLSP is $O(\frac{log(N)}{N})$ (overhead of DLSP-SN is $O(log(N))$ and that of GHLS is $O(N)$). For this reason, GLS exhibits a similar trend in Figure 4.13. However, the trend is not clear for DLSP, which can be explained as follows. Because of the network boundary, the number of location servers at any level increases as the network size grows. For example, the average number of level-0 (level-1) location servers increases from 4 to 6.25 (from 0 to 3) as N changes from 100 to 400. So, the trend is offset by the increase of overhead due to additional location servers.

In Figure 4.13, the overhead of GLS increases almost linearly at low speeds for the following reason. GLS does not use any timeout for sending updates, so its update overhead always increases linearly with the mobile's speed. In GHLS, the timeout is 8s and the movement threshold is 100m, and hence, at low speeds, the mobile sends location updates every 8s, and the overhead of GHLS is constant even when the mobile's speed increases. Therefore, the normalized overhead of GLS increases linearly at low speeds. Let's compare GLS with DLSP-SN. The overhead of DLSP-SN is at least 75% less than that of GLS, since it updates less location servers at each level and incurs less updates when the mobile crosses a square boundary. Let's compare GLS with DLSP in Figures 4.13 and 4.11. GLS is shown to incur a much higher overhead than DLSP for $400 \times 400m^2$ networks, because GLS updates the same number of level-0 location servers (4) as DLSP does for

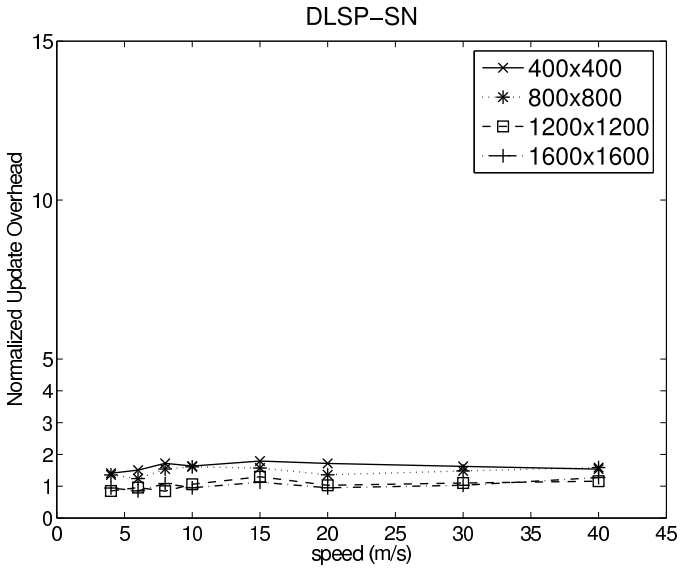


Figure 4.12: DLSP-SN reduces the update overhead by 70% or more. Its overhead is comparable to that of GHLS in a network of 900 nodes or more.

this network size, and it incurs more overhead in case of boundary-crossing. As the network size grows, DLSP selects more location servers than GLS, so its overhead catches up with or exceeds that of GLS.

Query-Path Length

The results plotted in Figure 4.14 are also from the same tests for the query-delivery ratio. Due to the gridding effect, the query-path length of DLSP-SN is $40-45\%$ longer than that of DLSP in large networks.

In Figure 4.14, the query-path length of GHLS decreases sharply beyond the mobile's speed of 10m/s, because more than 30% of queries (most of them take long paths) are dropped and thus not counted. Similarly, the query-path lengths of DLSP and GLS decrease noticeably at 30 and 40m/s. These speeds are consistent with Figures 4.7, 4.9 and 4.10. DLSP-SN results in longer query-paths than GLS, because DLSP-SN uses less location servers than GLS. So, DLSP-SN suffers more from the gridding effect. The results of smaller networks show the same trend with smaller gaps.

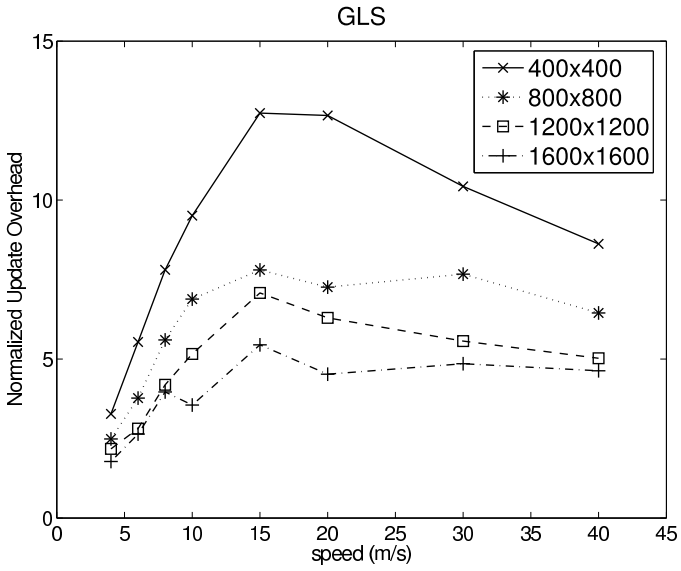


Figure 4.13: GLS incurs a very high update overhead because each level has 3 location servers, and boundary-crossing incurs additional overhead.

Adaptation

To evaluate how DLSP-ASN improves overall energy-efficiency over DLSP-SN, we use the same deployments and movements as in previous tests, but change the workload such that only two sensor nodes in the same level-0 square send data to the mobile at the same constant rate.

Figures 4.15 and 4.16 show the results from a network of 1600 nodes and the data rate of 0.5 message per 1 second for each node.

Figure 4.15 shows that, with adaptive location updates, DLSP-ASN has a slightly higher delivery ratio than DLSP-SN, because, in the first round, each query travels less hops in DLSP-ASN than in DLSP-SN. So it takes less time for a query to reach the mobile's known location, and thus, the mobile moved away less from the location. Starting at the second round, the query has a higher chance to be delivered successfully. Figure 4.16 shows the total energy cost normalized by the total cost of GHLS. The adaptive mechanism can improve the overall energy-efficiency by as much as 40%. For the range of 4 to 15m/s, the total energy cost of DLSP-ASN is comparable to, or even lower than, that of GHLS. This is because most queries are delivered in one round. During the first round, a query obtains the mobile's location information within a small square in DLSP-ASN, but in GHLS, it has to travel more hops to a randomly-picked location server within the largest square.

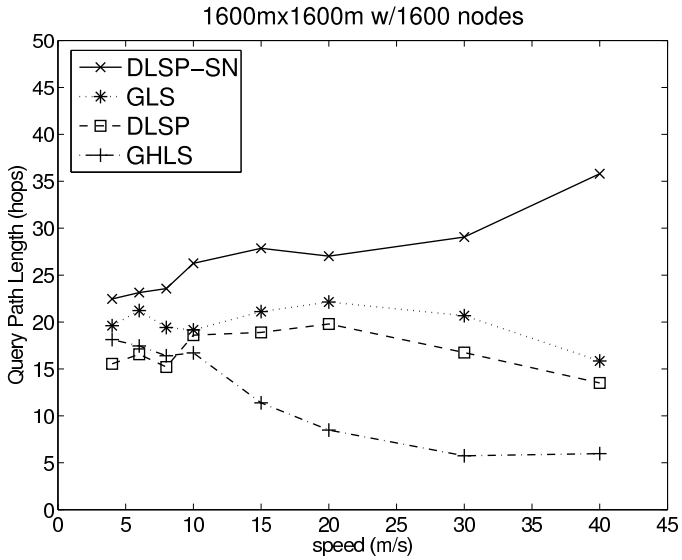


Figure 4.14: DLSP-SN has longer query paths due to gridding effect.

As the speed increases, the improvement diminishes because DLSP-ASN also needs to take more rounds to deliver the queries. In the tests of smaller networks and lower data rates, DLSP-ASN shows less improvement on overall efficiency as expected.

4.6.3 Simulation Results Using S-MAC

In order to validate our analysis of DLSP on a MAC with real low-power duty cycling, we simulated DLSP using S-MAC [79]. S-MAC is inspired by PAMAS [89] and uses RTS-CTS to reduce channel contention. S-MAC periodically sleeps, wakes up to receive and transmit packets if any, and then returns to sleep. Each active period is a constant period, and thus, the length of the sleep period determines the length of each frame (i.e., the active period plus the sleep period). To match the per-hop latency with our 802.11 MAC-based simulation, we set the sleep period such that the length of S-MAC duty cycle is 0.5s. Other parameters of S-MAC such as slot time are kept as default, resulting in 20.8% duty cycle. This duty cycle is rather high, but we keep the default parameters of S-MAC while setting the per-hop delay as 0.5s for a fair comparison.

Moreover, because S-MAC is designed for static sensor networks, we need to slightly modify it to deal with mobility — if a sensor node receives a beacon from a mobile, it stays awake for a short period (in our simulation, this period is 2.5s). With this modification, the neighbor sensors of the mobile are given enough wakeup time to send their replies and avoid severe collision, which could

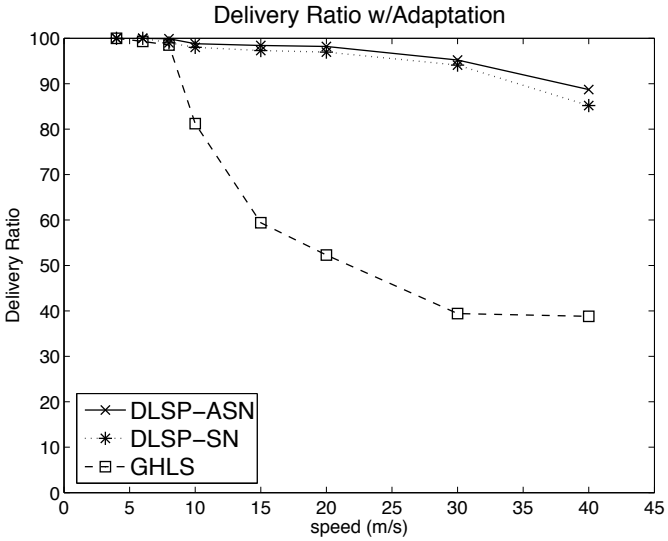


Figure 4.15: The delivery ratios of DLSP and GHLS match the results in previous figures.

occur without the modification. The mobile sink does not observe S-MAC sleep-scheduling, so it can communicate with sensor nodes with different sleep schedules. This may be justified because the mobile is usually much less power-constrained than sensors.

If a packet is lost, the sender tries to send it again (up to 3 times) in the next active period. Therefore, the actual per-hop latency varies with sleep scheduling and packet retransmission. The results presented next are from the test runs using same deployments and movements as in the previous simulation.

Unlike the DLSP simulation with 802.11 MAC with fixed delay of 0.5s, DLSP with S-MAC can have a delay shorter than its frame length (which is set to 0.5s) for the following reasons. First, the sensors surrounding the mobile sink stay awake longer than other sensor nodes, so the latency between these sensors and the mobile is much less than that between two sensors under low-power duty cycling, which is approximately 0.5s. Second, at the origin of each packet, the per-hop latency is on average, a half of duty cycle. As the network size increases, location updates and queries take more sensor-to-sensor hops, so the average per-hop latency increases. The mean and standard deviation of the per-hop latency, and the speed limit derived from Eq. 4.13 are shown in Table 4.3.

Figures 4.17–4.19 plot the query-delivery ratio of DLSP, DLSP-SN, and GHLS, respectively, using S-MAC. With S-MAC, the performance of DLSP protocols (DLSP and DLSP-SN) also

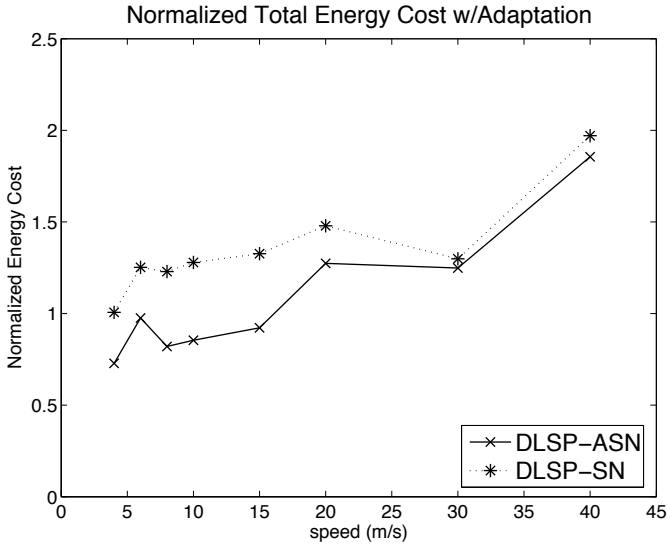


Figure 4.16: The energy cost of DLSP-ASN is even less than GHLS when the speed is below 15m/s, when both provide high packet-delivery ratios.

validates our analysis — DLSP protocols scale well if the mobile's speed is under the limit as shown in Table 4.3, and degrade as the mobile's speed grows beyond the movement threshold. Compared to Figures 4.7, 4.17 shows that the performance of DLSP with S-MAC is slightly better than that of DLSP with 802.11-MAC, while the performance of DLSP exhibits similar trends regardless of the underlying MAC protocol. This performance improvement is attributed to the fact that the per-hop latency with S-MAC is below 0.5s.

4.6.4 Results with Additional Mobility Models

We also evaluated DLSP with two additional mobility models: random walk mobility model and Gauss-Markov mobility model [88]. The random walk model has strong randomness as in the random way-point model. However, in the random walk model, mobile nodes randomly choose their speed and direction at each fixed interval instead of randomly choosing the destination in the random way-point model. In the Gauss-Markov mobility model, the speed and direction of mobile nodes are correlated over time and modeled as a Gauss-Markov stochastic process. The temporal-dependency of the Gauss-Markov model is determined by two parameters: σ is the standard deviation of the randomly-generated speed and direction at each interval, and α is the memory-level

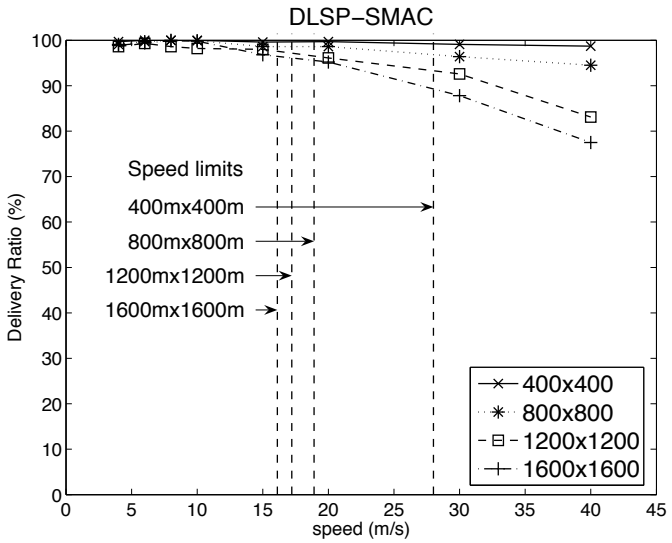


Figure 4.17: The query-delivery ratio of DLSP with S-MAC in a 1600m×1600m network. DLSP with S-MAC scales well if the mobile's speed is below the threshold shown in Table tab:dlsp-smac-delay.

parameter which determined the randomness of the mobility model. The details of these mobility models can be found in the survey paper [88].

Figures 4.20 and 4.21 plot the performance of DLSP with different mobility models. Here we present a subset of our test results to show the effect of different mobility models with different parameters. For the random walk model, the interval duration is set to 20s and 40s and the interval duration is set to 10s to 20s with $\alpha = 0.5$ and $\sigma = \pi/2$ for the Gauss-Markov model. The mobile's speed ranges from 4 to 40m/s as in our previous simulation. It is shown in Figures 4.20 and 4.21 that DLSP scales well if the mobile's speed is under the threshold with any mobility model, which confirms our analysis.

However, at a high speed, it is observed that the degree of performance degradation differs from one mobility model to another, and from one configuration to another, because the performance of location services is affected by other parameters of mobility models, even if the (average) speed is the same. For example, the duration of interval of the random walk model affects the performance of the protocol; the longer the duration of the interval, the further the mobile moves. Then, location queries that have been forwarded to the mobile's previous location will be less likely to be in the proximity of the mobile, and the delivery ratio may degrade. This effect is more pronounced when

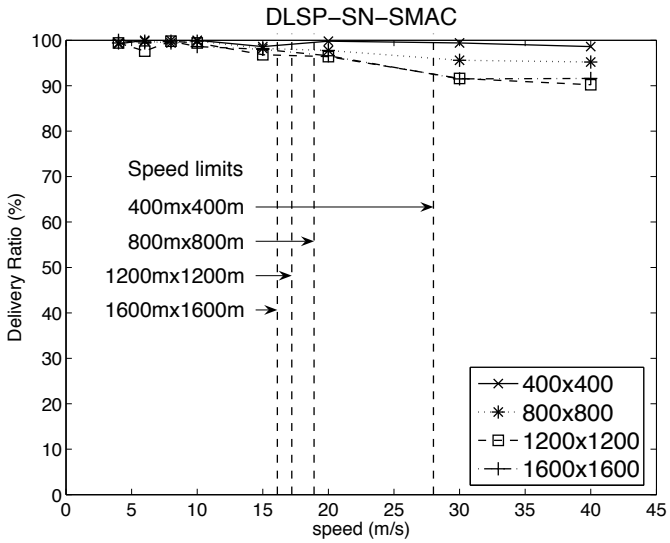


Figure 4.18: The query-delivery ratio of DLSP-SN with S-MAC in a 1600m×1600m network. DLSP-SN with S-MAC also scales well if the mobile's speed is below the movement threshold.

a mobile moves at a high speed in a large network, in which it hits the network boundary less frequently. For example, in Figure 4.21, it is shown that the random walk mobility model with a 20s duration (RW20) outperforms the same model with a 40s duration (RW40) at 40m/s. So is the Gauss-Markov mobility.

There are previous efforts on developing the common metrics that can interpret the parameters of different mobility models [90, 91], but not a single good metric has been developed yet, and most of them focused on routing protocols, not on location services. Development of a universal measure of mobility models for location services is beyond the scope of this chapter.

4.7 Concluding Remarks

In this chapter, we presented a distributed location service protocol (DLSP) which considers the non-trivial per-hop latency caused by low-power duty cycling in sensor networks. Through a rigorous analysis of DLSP, we derive the condition for achieving a high packet-delivery ratio, and show how to configure the protocol parameters to ensure the scalability of DLSP. DLSP is shown to be scalable if the mobile's speed is below a certain fraction of the packet-transmission speed, which depends on a movement threshold. The theoretical mobile's speed limit is a one-fifth of the

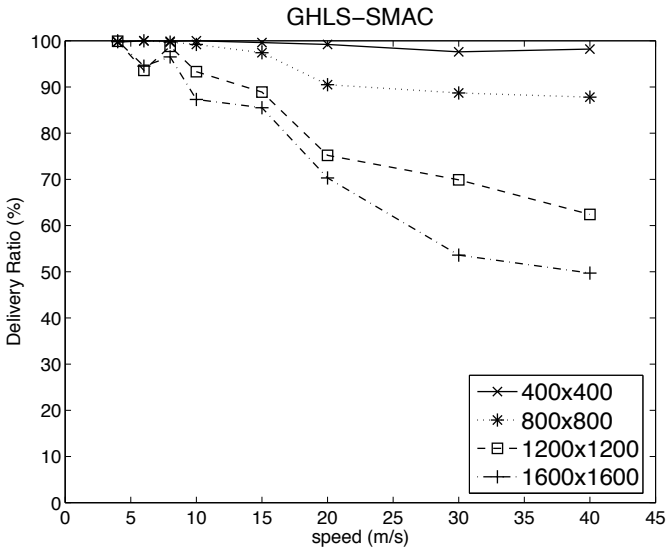


Figure 4.19: GHLS with S-MAC shows a similar trend as GHLS with 802.11 MAC. Since there is no single speed limit for different network sizes in GHLS, the performance degrades even at a lower speed for large networks.

packet-transmission speed.

We evaluate DLSP with extensive simulation using two MAC protocols and three mobility models. The evaluation results confirm our analysis. We also proposed an optimization technique, DLSP-SN, that can reduce the location-update overhead by 70% or more, while its query-delivery ratio is even better than DLSP when the mobile's speed is high. In order to make a tradeoff between update and data-delivery costs, we presented a greedy adaptation mechanism, DLSP-ASN, which can significantly improve overall energy-efficiency.

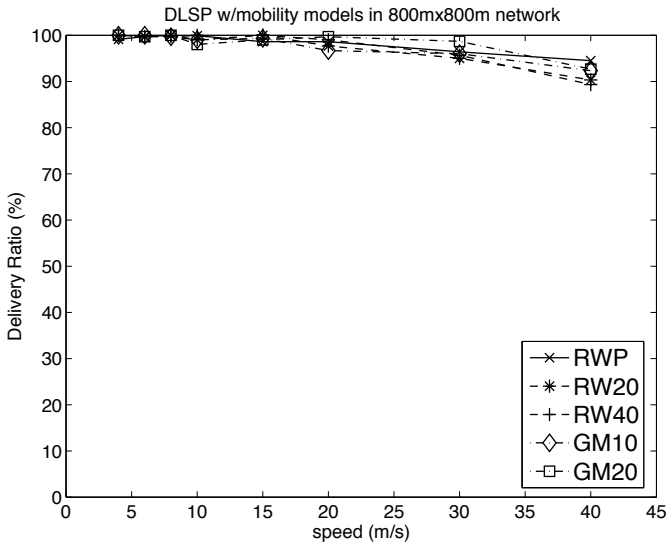


Figure 4.20: The random way-point model (RWP), the random walk model (RWP) with the duration of 20s (RW20) and 40s (RW40), and the Gauss-Markov model with the duration of 10s (GM10) and 20s (GM20) are simulated with DLSP in an 800m$\times$800m network.

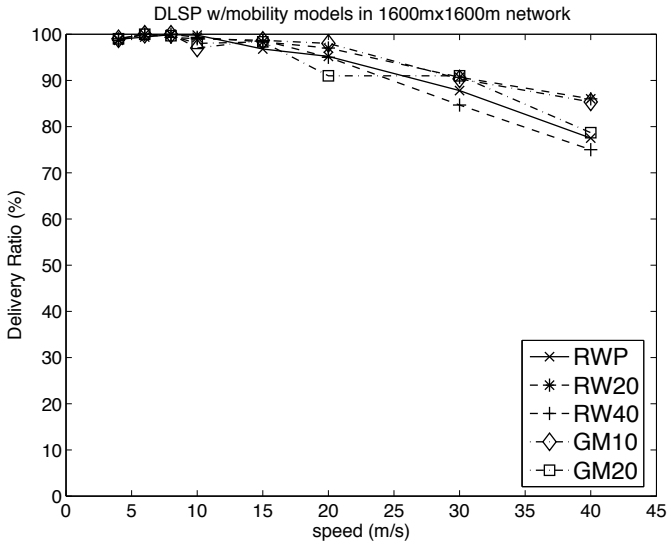


Figure 4.21: The random way-point model (RWP), the random walk model (RWP) with the duration of 20s (RW20) and 40s (RW40), and the Gauss-Markov model with the duration of 10s (GM10) and 20s (GM20) are simulated with DLSP in an 800m$\times$800m network.

$P(S), P(R,T)$	Location of a stationary sensor node S, or of a mobile node R at time T
$S_{k,j}(S)$, $S_{k,j}(R,T)$	$S_{k,0}(S)$ is the level-k square the sensor S resides, and $S_{k,0}(S)$ ($j = 1, \cdots, 8$) are the eight level-k neighbor squares adjacent to $S_{k,0}(S)$. $S_{k,j}(R,T)$ is the level-k square the mobile R resides at time T, and $S_{k,j}(R,T)$ ($j = 1, \cdots, 8$) are the level-k neighbor squares
$LS_{k,j}(R,S)$, $LS_{k,j}(R,T)$	R's level-k location server in the square $S_{k,j}(S)$, or $S_{k,j}(R,T)$
h	Level of the largest square, i.e., the entire deployment field
ℓ	Edge length of a level-0 square
m	Movement threshold for level-0 location servers $2^{-m}\ell$
τ	Time threshold for location updates at level-0 location servers
t_h	Average per-hop latency, including transmission/retransmission time, and scheduling delay
p	Average per-hop progress, or decrease of Euclidean distance to the destination for each hop taken
r	Radio range
$\bar{v}$	Mobiles' average speed
$dist(P_1, P_2)$	Distance between two locations, P_1 and P_2

Table 4.1: List of symbols

Protocols	DLSP	DLSP-SN	DLSP-ASN	GLS
ID-to-LS mapping	Hash the mobile node ID into a location within a square, and the node closest to the location becomes a location server	Same as DLSP	Same as DLSP	Select the node whose ID is closest to the mobile node ID within a square
LS in neighbor squares	One LS in each of the eight neighbor squares	At most one location server from a neighbor square	Same as DLSP-SN	One LS from each of the three neighbor squares
Cross-boundary updates	Location updates can only be triggered by timeouts or exceeding the movement threshold	Same as DLSP, but the selected neighbor depends on whether or not a boundary of certain level is crossed	Same as DLSP-SN	Need to update the three neighbor location servers[3] from level-0 to level-k if the boundary of a level-k square is crossed
Data source adaptiveness	No	No	Yes A mobile may send additional location updates to the location servers near the data sources according to a greedy algorithm	No. A mobile node may send its location to the data sources at a fixed rate
Griding effect[4]	No	Yes	Not after the additional location updates are sent out	Yes
Multi-rounds	Yes, with restrictions (1-3)[5]	Yes, with restrictions (1-2)[5]	Same as DLSP-SN	No

Table 4.2: Comparison of hierarchical location services

Network size	Average per-hop latency	Standard deviation	Movement threshold
400m $\times$ 400m	0.251s	0.271	27.9m/s
800m $\times$ 800m	0.373s	0.281	18.8m/s
1200m $\times$ 1200m	0.410s	0.280	17.1m/s
1600m $\times$ 1600m	0.439s	0.313	16.0m/s

Table 4.3: Average per-hop latency of DLSP using S-MAC. The average per-hop latency and its standard deviation vary with the network size. The movement thresholds can thus be derived from our analysis using Eq. 4.13 for different network sizes.

CHAPTER 5

Location Information Scrambler: Privacy Protection for Smartphone Users

5.1 Introduction

As GPS-integrated smartphones, such as Apple's iPhone 3G and Google's G1, are increasingly popular, location-based services (LBSs) are becoming a new buzz in the IT industry, attracting significant attention from mass media [92, 93], financial investors [94], companies [95, 96, 97, 98], and potential customers. LBS is an information or entertainment service that exploits knowledge about users' geographic locations to deliver highly personalized information tailored to the users' needs, for example, restaurants, gas stations, and other points of interest (POIs) within certain proximity of the mobile users. According to a report by Computer Science and Telecommunications Board [99], LBSs are expected to be seamlessly and ubiquitously integrated into future computing environments and users' daily lives.

However, with the growing deployment and use of LBSs, mobile users are increasingly enticed to reveal their location information, thus becoming vulnerable to malicious attacks on their location privacy. This is mainly because the location information is usually communicated via open wireless networks and stored in the LBS providers. Adversaries can acquire unauthorized access to users' location information by eavesdropping on the communications between users and servers, or compromising the location servers. In general, there are two types of location privacy attacks: *location disclosure* [100, 101] and *movement tracking* [102, 103]. Location-disclosure attacks occur when attackers utilize users' location information to infer a wide range of the users' personal information, such as home location, personal habits/preferences, and medical conditions,

for unwanted and/or malicious purposes (e.g., unsolicited advertisements). Movement-tracking attacks occur when attackers infer a user's daily routes and future locations, which may even cause physical harassments to the user, such as those in stalking or illegal surveillance [104].

To protect users' location privacy, two types of approaches have been proposed. The first type introduces the uncertainty of user identity (ID) by cloaking location information [24, 25]. Specifically, a user's location, masked by a geographic area and a time window, is *k-anonymous* if and only if this user and at least $k-1$ other users were present in the area during the time window. So, attackers cannot distinguish among k or more users. The second type introduces "unlinkability" between different pseudonyms of a user by using *mix zones* [26]. A mix zone is defined as an area where user IDs are "mixed" and users change to new, unused pseudonyms whenever they enter a mix zone. This way attackers cannot link people going into the mix zone with those coming out of it. Both approaches require a trustworthy third-party server so that they can count the number of users inside a cloaked or mix zone and ensure that the uncertainty of user IDs or the degree of unlinkability between pseudonyms is met.

In this chapter, we propose a new approach, called the *Location Information ScrAmbler* (LISA), to protecting the location privacy of mobile users, using *m-unobservability*. The key idea behind LISA is to disable the distinguishability of each user's POIs, and therefore, weaken the attackers' capability of inferring the user's private information or mobility patterns from his locations. LISA achieves *m-unobservability* by intentionally introducing measurement noise into the location information provided to a location server, such that the users' locations estimated by attackers are *m-unobservable*—at least m POIs can be related to each estimated location. By using a simple yet general object tracking model based on an extended Kalman filter [105], LISA adjusts the location (measurement) noise to obtain the covariances of location estimation that meet *m-unobservability*.

Since LISA performs such "noise-level tuning" locally on individual mobile devices, it eliminates the reliance on trustworthy third-party servers required by other approaches. In LISA, mobile users need to trust only their handheld devices and can set up personalized privacy requirements. This simplifies trust relationships and improves configuration flexibility, thereby significantly reducing the complexity in the design of location servers and their deployment. In contrast, the trust relationship between third-party servers and mobile users in previous approaches is often established through policy restrictions, making no guarantees on the protection of users' location

privacy against malicious attacks.

The advantages of LISA are three-fold as follows.

- It introduces a new, orthogonal dimension of uncertainty and can be combined with existing approaches to provide stronger location privacy protection.
- It prevents leakage of mobile users' privacy information as a result of compromised third-party servers, and limits privacy attacks to individual cell phones, thus significantly reducing the impact of attacks.
- It lowers the trust requirements from mobile users, thus simplifying the implementation and deployment of LBSs.

The chapter is organized as follows. Section 5.2 describes our threat model, and Section 5.3 introduces the privacy model and metrics used in LISA. Section 5.4 presents the details of LISA, and Section 5.5 describes two optimization techniques for performance improvement. Section 5.6 discusses our evaluation methodology and results. Section 5.7 compares LISA with other related work. Section 5.8 discusses future work and concludes the chapter.

5.2 System and Threat Model

We assume that an LBS system consists of smartphones, wireless networks, a proxy server, and LBS servers, as shown in Fig. 5.1. A smartphone user who wants to access a location-based service sends a service query to the proxy server through a wireless network, such as AT&T and T-Mobile. The service query includes information about the user's current location, obtained through an integrated GPS or triangulation of nearby radio towers, and the types of service he/she likes to access, so that the proxy server can pass it to appropriate LBS servers. The LBS servers then return the responses to the user through the proxy server. The communication between the mobile device and the proxy server goes through encrypted connections to prevent unauthorized eavesdropping.

Here we adopt a threat model in which users do not need replies from trustworthy third-party servers (such as the location server [25] or the anonymity server [24]) to protect location privacy. LISA assumes that a user's own handset is trustworthy, and neither the proxy server nor the LBS servers are trusted. Hence, adversaries may compromise the untrustworthy hosts to access *any*

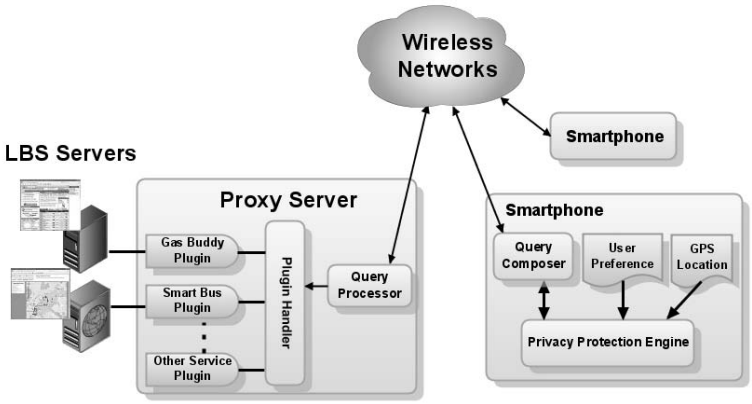


Figure 5.1: The architecture of a typical LBS system

information in users' LBS service queries, such as their IP addresses, current locations, and past trajectories. This model removes the dependency on a trusted entity and is more realistic and attractive in that (1) it is very expensive and difficult, if not impossible, to perfectly secure a public server, and (2) many data breaches occur due to compromised servers, dishonest insiders, and accidental loss or improper disposal of storage media.

In addition, as in previous work [25, 24, 106], we also assume adversaries only read the location information in the compromised servers and do not attempt to manipulate the query results sent from LBS servers to mobile users. This is because attackers' main purpose is to learn users' location privacy information. Modifying or faking query responses not only provides few or no benefits for attackers to achieve the goal but also add much risk of getting detected, as users can verify the server responses with their observation.

Third, we assume that the adversaries use the source IP addresses in the service queries to track the users' movement. Although IP addresses assigned to each cell phone are usually dynamically allocated by wireless carriers, our experiments demonstrated that the IP address assigned to a user often remains unchanged for an extended period of time and IP collision, i.e. different users get assigned the same IP address, rarely happens, as shown in Fig. 5.2.

The data in Fig. 5.2 was collected from the experiment we ran from 9:00AM to 2:30PM on a normal workday using 2 T-Mobile G1s phones. In this experiment, one cellphone initiated a TCP connection through the T-Mobile EDGE network to a server every 1 minute, while the other

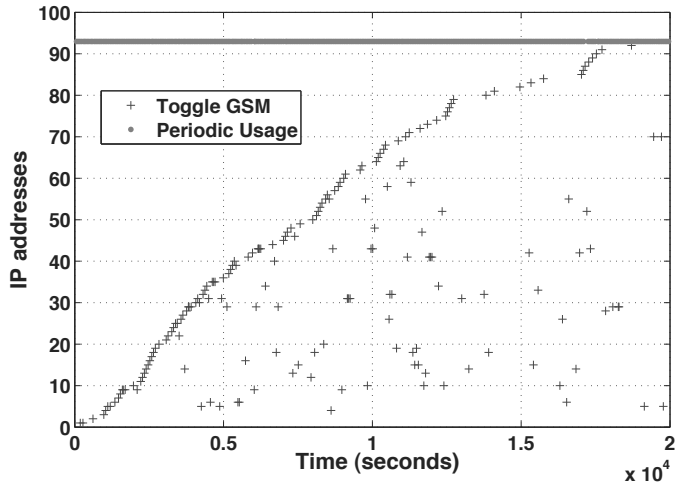


Figure 5.2: The IP addresses T-Mobile assigned to two G1 phones. The X-axis indicates the time an IP was assigned, and the Y-axis indicates different IPs.

turned off the radio, then turned it back on and made a TCP connection every 1 minute in order to force an IP re-assignment from the carrier network, T-Mobile. The IP addresses of both phones are depicted in Fig. 5.2. It shows that the first G1 phone always acquired the same external IP from T-Mobile, as indicated by the straight line (labeled as "Periodic Usage"). the second G1 phone was never given the first phone's IP, as shown by the data points of "Toggle GSM." Each point along the contour curve means a new IP assigned by T-Mobile, and each point below the contour represents the reuse of a previous IP.

Based on this experimental result, attackers may very likely identify a sequence of service queries from a user and construct a trajectory which potentially can leak more privacy information.

5.3 Privacy Model and Metrics

A survey by Pfitzmann and Koehntopp[107] generalizes and formalizes the terminology on anonymity and related topics as follows.

- "Anonymity is the state of being not identifiable within a set of subjects, the anonymity set"
- "Unlinkability of two or more items (e.g., subjects, messages, events, actions, ...) means

that within this system, these items are no more and no less related than they are related concerning the *a priori* knowledge."

- "Unobservability is the state of IOIs being indistinguishable from any IOI at all", where IOI stands for item of interest.

Using this terminology, Gruteser and Grunwald [25] proposed *k-anonymity*, which quantifies location privacy as the uncertainty of user identity. Specifically, a user's location information, specified by a geographic area and a time period, is said to be *k-anonymous* if and only if this user and at least $k-1$ other users were present in the area during the given time period. Beresford and Stajano [26] proposed mix zones to achieve unlinkability between different pseudonyms of a user. Users change to new, unused pseudo names whenever they enter a mix zone, so that attackers cannot link people going into the mix zone with those coming out of it.

Following the same terminology, we propose *m-unobservability* to quantify the uncertainty of a user's location. Items of Interest (IOIs) can be assigned various meanings, such as point of interests (POIs) in the context of location-based services. We define that the estimation of a user's location, specified by a mean vector (μ) and a covariance (Σ), is *m-unobservable* if and only if it can be related to at least m points of interest. Specifically, assuming $\mu = (\bar{x}, \bar{y})$ and

$$\Sigma = \begin{pmatrix} \delta_x^2 & \delta_{xy} \\ \delta_{yx} & \delta_y^2 \end{pmatrix}$$

, and $f(x,y)$ is the probability density function of the location estimation

$$\|P_i \mid \text{POI } P_i \text{ falls into the area } D\| \geq m \tag{5.1}$$

where $D = \{(x,y) \mid |x-\bar{x}| < \alpha * \delta_x \text{and} |y-\bar{y}| < \alpha * \delta_y \}$ and α satisfies $\iint_D f(x,y)\,dxdy \geq 0.9$.

The intuition behind the *m-unobservability* is that if there are at least m POIs in the proximity (restricted by probability 0.9) of the mean estimated location, attackers cannot relate any particular POI to the estimated location, and thus, their ability to infer the user's private information is significantly weakened. For example, a bus rider may provide his location to a real-time campus bus schedule system in order to obtain the bus arrival times at nearby bus stops, and therefore, all campus buildings and bus stops are considered as POIs. If the rider's location information can be

related to m different buildings or stops, an attacker will not be certain of which building or stop the rider visits.

Moreover, the set of POIs that supports *m-unobservability* depends on the application scenarios and users' needs. Using the above example, if a user is concerned about being stalked by strangers, only the bus stops are accounted for, so a potential stalker cannot determine to which bus stop the user goes.

The *m-unobservability* also provides a measure for location uncertainty—the larger m, the higher the degree of location uncertainty and location privacy. Because attackers may obtain a sequence of location service queries as the user moves around, our approach aims to protect the privacy of his location and movement trajectory. Therefore, our privacy metric is the average entropy of the sequence. Formally, for any user u,

$$\bar{H}^u = \frac{1}{n} \sum_{l_i \in L} H_i^u \tag{5.2}$$

and

$$H_i^u = -\sum_{j=1}^{m_i} p_j log(p_j) = log(m_i) \tag{5.3}$$

where $L = l_1, l_2, \cdots, l_n$ is the sequence of locations where u accesses some location services, and m_i is the number of POIs that attackers can relate to the estimated location at l_i. Eq. (5.3) is derived from the general privacy entropy definition [108] by assuming each of the m_i POIs in the proximity is equally likely to be related to the user, i.e., $p_i = 1/m_i$.

5.4 Protection of Location Privacy Using *m-unobservability*

This section details our approach. We first state our design goals (Section 5.4.1), then outline the architecture of a LBS system (Section 5.4.2), and finally, describe how to protect location privacy.

LBS	City	POI Id	POI Type	POI Name	Latitude	Longitude
GasBuddy	New York	1	GasStation	Mobil	40.8393	-73.9373
GasBuddy	New York	2	GasStation	BP	40.7249	-73.9958
GasBuddy	New York	3	GasStation	Sunoco	40.6724	-73.8429
GasBuddy	New York	4	GasStation	BP	40.5930	-73.9326
GasBuddy	Ann Arbor	1	GasStation	BP	42.2550	-83.6873
GasBuddy	Ann Arbor	2	GasStation	Shell	42.2565	-83.6962
GasBuddy	Ann Arbor	3	GasStation	Marathon	42.2847	-83.8022
SmartBus	Ann Arbor	1	Bldg	W Hospital	42.2827	-83.7290
SmartBus	Ann Arbor	2	Bldg	Crisler Arena	42.2650	-83.7467
SmartBus	Ann Arbor	1001	BusStop	CC Little	42.2781	-83.7350
SmartBus	Ann Arbor	1002	BusStop	Bursley Hall	42.2946	-83.7208

Figure 5.3: The POI database at the proxy server

5.4.1 Design Goals

Our approach should meet the following two goals.

- Privacy Protection: it must protect users' location privacy even when attackers compromised the proxy server and the LBS servers.
- Resource Constraints: the software running on smartphone clients must operate under various resource constraints, especially battery and processor capability constraints.

5.4.2 LBS Architecture

As mentioned in Section 5.2, an LBS system consists of smartphones, wireless networks, a proxy server, and LBS servers. Each LBS provider maintains a list of POIs and certain information about the POIs that users may query. For example, GasBuddy keeps a list of gas stations in a number of cities and up-to-date gas price at each station. SmartBus has a list of campus buildings and bus stops and the bus arrival time(s) at each stop. When an LBS provider subscribes to the system, it provides its POI list to the proxy server, so that the proxy can build a POI database, as shown in Fig. 5.3.

On each smartphone, Query Composer is responsible for sending location-service queries to, and receiving responses from the proxy server. Suppose a user wishes to query the gas prices near the current location (x,y), or specifically, in the rectangular area, $(x-L,y-L,x+L,y+L)$. If (x,y) is sent to the proxy and GasBuddy, the user's location privacy may be risked. Instead, the Privacy Protection Engine computes a scrambled location, $(\prime x,\prime y)$, such that it is *m-unobservable*, where m is determined by the user's privacy requirement. For example, the user may prefer that at least

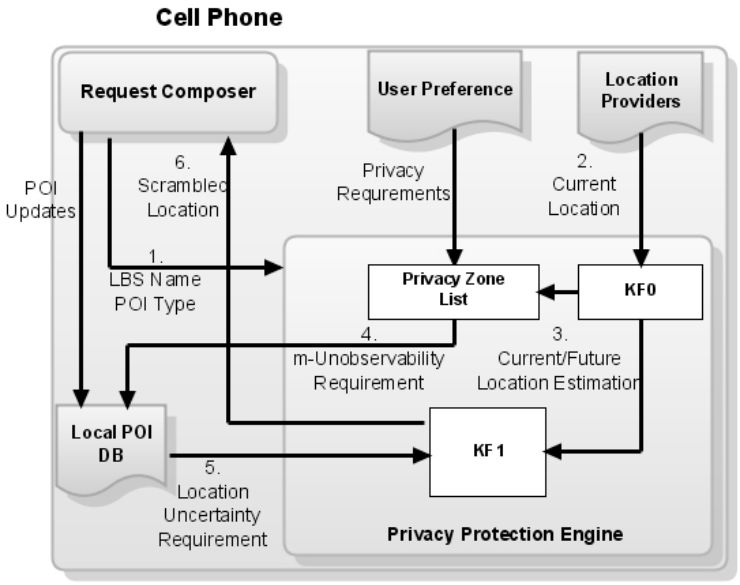


Figure 5.4: The Privacy Protection Engine

10 POIs can be related to the scrambled location. To count nearby POIs, each handset maintains a local POI database, by querying the proxy's POI database with certain constraints, such as the city where the user lives or visits.

With privacy protection, the location-service query contains a rectangle region $(\prime x - \alpha\delta_x - L, \prime x + \alpha\delta_x + L, \prime y - \alpha\delta_y - L, \prime y + \alpha\delta_y + L)$. Upon receiving queries from the client, the proxy server passes them to the plugin handler, which selects the right plugins to communicate with the corresponding LBS servers, and then returns the response to the client handset. The handset locally filters out the results outside the rectangular area $(x-L, y-L, x+L, y+L)$.

5.4.3 Location Privacy Protection Engine

Location information is scrambled by the Privacy Protection Engine, as detailed in Fig. 5.4. We first introduce a mobility model the Engine adopts, then present the three key data structures—Privacy Zone List, KF0 and KF1—and finally, describe how the Engine scrambles a user's location information.

Mobility Model

To track a user's movements, the Engine adopts the Wiener-sequence acceleration model [109] which assumes each acceleration increment is an independent (white noise) process. Let x_k denote the state variable and y_k the observation (a.k.a. measurement) variable at time t_k. In fact, the process state, x_k, is a vector in the form of $(x, v_x, a_x, y, v_y, a_y)\prime$, which represents the location, velocity, and acceleration of a user on the X and Y axes. The mobility model is given by

$$x_{k+1} = A_k(\Delta t_k)\, x_k + G(\Delta t_k)\, w_k \tag{5.4}$$

$$y_k = C\, x_k + v_k \tag{5.5}$$

where

$$A_k(\Delta t_k) = \begin{pmatrix} 1 & \Delta t_k & \Delta t_k^2/2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t_k & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t_k & \Delta t_k^2/2 \\ 0 & 0 & 0 & & 1 & \Delta t_k \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$G_k(\Delta t_k) = \begin{pmatrix} \Delta t_k^2/2 \\ \Delta t_k \\ 1 \\ \Delta t_k^2/2 \\ \Delta t_k \\ 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

and w_k are both random variables for the process (system) noise and the observation (measurement) noise, respectively, and $\Delta t_k = t_{k+1} - t_k$. Therefore, a Kalman filter[1] [105] based on the mobility

[1]The filter is sometimes called Stratonovich-Kalman-Bucy filter because it is a special case of a more general, non-linear filter developed earlier by Ruslan L. Stratonovich.

model is summarized as follows. The predicted process state is:

$$\hat{x}_{k+1|k} = A_k(\Delta t_k)\,\hat{x}_k \tag{5.6}$$

and the predicted estimate covariance is:

$$P_{k+1|k} = A_k(\Delta t_k)\,P_{k|k}\,A_k(\Delta t_k)\prime + Q_k \tag{5.7}$$

where the process noise covariance is:

$$Q_k(\Delta t_k) = cov(G_k(\Delta t_k)w_k) =$$

$$var(w_k)\begin{pmatrix} \Delta t_k^4/4 & \Delta t_k^3/2 & \Delta t_k^2/2 & 0 & 0 & 0 \\ \Delta t_k^3/2 & \Delta t_k^2 & \Delta t_k & 0 & 0 & 0 \\ \Delta t_k^2/2 & \Delta t_k & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta t_k^4/4 & \Delta t_k^3/2 & \Delta t_k^2/2 \\ 0 & 0 & 0 & \Delta t_k^3/2 & \Delta t_k^2 & \Delta t_k \\ 0 & 0 & 0 & \Delta t_k^2/2 & \Delta t_k & 1 \end{pmatrix}$$

.

For measurement updates, the Kalman gain is:

$$K_k = P_{k+1|k}C\prime[C\,P_{k+1|k}C\prime + R_{k+1}]^{-1}$$

, the updated state estimate is:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}[y_{k+1} - C\,\hat{x}_{k+1|k}$$

, and the updated estimate covariance is:

$$P_{k+1|k+1} = [I_6 - K_{k+1}\,C]\,P_{k+1|k}. \tag{5.8}$$

Next, we will discuss how this model is used in the description of KF0 and KF1.

Three Key Data Structures

The Privacy Zone List keeps a number of places where a user's location privacy needs to be protected, because protecting location privacy consumes extra resources, particularly battery power

of the handset, and therefore, a user may want to specify where and how his location privacy should be protected. For example, based on the user's preference, the list may include the user's home location and the hospital he often visits. Moreover, the degree of privacy protection may vary from one place to another, such as *10-unobservability* for home and *5-unobservability* for the hospital. The engine can then look up the local POI database, and find a *privacy zone* that covers the home (hospital) location and includes 10 (5) *relevant* POIs. Here *relevant* POIs are meant to confuse attackers. For example, only apartments and residence buildings should be counted for protecting the privacy of home.

KF0 denotes the extended Kalman filter that the Engine uses to track the user's movement. Such tracking is necessary for the following two reasons. First, location measurements through radio tower triangulation is inaccurate, and the filter can be used to reduce measurement errors. Triangulation is still useful for smartphones without an integrated GPS (A-GPS), such as the first-generation iPhone, and for saving battery power even in phones with GPS [110]. Second, the Engine needs to predict the user's movement (using Eq. (5.6)) to improve privacy protection and reduce energy consumption. The look-ahead optimization based on movement prediction is detailed in Section 5.5.

KF1 denotes the extended Kalman filter that the Engine uses to scramble the user's accurate location information. Given a location, (x,y), a scrambled location, $(x\prime,y\prime)$ is a random variable following the normal distribution with mean $\mu = (x,y)$ and covariance Σ. In fact, Σ is the covariance of the measurement noise that the Engine injects to confuse attackers. Then, from the attacker's perspective, a scrambled location is a location measurement with some significant measurement noise, which can be filtered or reduced using some mobility model. Hence, the role of KF1 is to examine a scrambled location and determine how much noise is needed to achieve the location uncertainty (given by Eq. (5.8)) that satisfies the privacy requirement.

Location Scrambling

The Privacy Protection Engine scrambles the user's location information in six steps, as illustrated in Fig. 5.4.

S1. When the user wishes to access an LBS, the Request Composer tells the Engine the LBS's name and the types of POIs that support unobservability.

S2. The Engine then gets a location measurement from one of the location information providers, such as A-GPS or radio tower triangulation. Assuming that the measurement is timestamped at t_k, the engine updates the parameter matrices of KF0 and KF1 using $\Delta t_k = t_k - t_k - 1$, and uses KF0 to estimate the current location (x_k, y_k).

S3. With the current location (x_k, y_k), the Engine searches the Privacy Zone List, and verifies if the location is inside any of the privacy zones derived from the user's preference. If so, the Engine gets the privacy requirement in the form of *m-unobservability*. If the location within multiple zones, the Engine gets the largest m.

S4. With m, the Engine looks up the local POI database and finds the rectangular region, $(x_k - d, y_k - d, x_k + d, y_k + d)$, such that there are at least m POIs of the specified types inside the region.

S5. With d, the Engine gets $\delta_x = \delta_y = d/2$. Here $\alpha = 2$ because, using the Kalman filter KF1, the location estimation follows a normal distribution with $\mu = (x_k, y_k)$ and

$$\Sigma = \begin{pmatrix} \delta_x^2 & \delta_{xy} \\ \delta_{yx} & \delta_y^2 \end{pmatrix}$$

. We have

$$\int_{x_k - 2\delta_x}^{x_k + 2\delta_x} \int_{y_k - 2\delta_y}^{y_k + 2\delta_y} f(x,y)\, dxdy \geq 0.9$$

, where $f(x,y)$ denotes the pdf of the normal distribution.

S6. Let $R_k = r I_2$. The Engine adjusts r such that $P_k(1,1) \geq d^2/4$ and $P_k(3,3) \geq d^2/4$. Then, it generates a random noise (e_x, e_y) (i.e., a location offset) that follows a normal distribution $((0,0)\prime, r^2 I_2$. Therefore, the scrambled location sent to the Request Composer is $(x_k + e_x, y_k + e_y)$ if $e_x, e_y \leq 2r$, or $(x_k + 2r, y_k + 2r)$ otherwise.

5.5 Optimizations

In this section, we propose two optimizations, look-ahead and de-randomization, to improve location privacy protection.

5.5.1 Look-ahead

In the previous section, we showed that KF1's state transition matrix, A_k, and process noise covariance, Q_k, are both determined by the inter-arrival time between consecutive LBS queries, Δt_k. Specifically, as Δt_k gets smaller, A_k and Q_k both become smaller. As a result, the prior estimation error covariance $P_{k+1|k}$ approaches 0 and so does the Kalman gain, K_k. Then, the location measurement is trusted less, while the predicted location is trusted more. In this case, if $P_{k|k}$ is also small (i.e., not in any private zone), increasing R_{k+1} cannot effectively increase $P_{k+1|k+1}$, and thus cannot satisfy *m-unobservability*.

The look-ahead optimization alleviates this problem by predicting the location privacy requirement in the next step and adjusting the measurement noise in the current step to meet the requirements in both steps.

First, the Engine needs to predict the location privacy requirement in the next step. If the inter-arrival times of LBS queries, $\Delta t_k (k = 1, 2, \cdots)$, are fixed at Δt, replacing Δt_k with Δt in Eq. (5.6) leads to the predicted location, which is then used by the Engine to search the Privacy Zone List and obtain the privacy requirement.

If the inter-arrival times follow some distribution (e.g., exponential distribution) with $f(t)$ as its pdf, then the next location $\hat{x}_{k+1|k} = A(t)\hat{x}_k$ with probability $f(t)$. Therefore, the mean privacy requirement is given by

$$\frac{\int_0^{t_0} p(t) f(t)\, dt}{\int_0^{t_0} f(t)\, dt}$$

, where $p(t)$ is the privacy requirement at the predicted location $A(t)\hat{x}_k$, and t_0 specifies how far in future the prediction goes. Obviously, t_0 should not be too large because the prediction can go very wrong.

Next, the Engine searches for the measurement noise covariance for this step and the next step, R_{k+1}, $R-k+2$, that satisfies both the privacy requirement of this step and the predicted requirement of the next step, such that $R_{k+1} + R - k + 2$ is minimized.

5.5.2 De-randomization

Another issue about the model in the previous section is that it does not consider movement between two LBS queries. If a user moves very little in a private zone but keeps sending LBS queries, his true locations may be disclosed even though the queries only contain scrambled locations. The reason is simple: if a sequence of scrambled locations are based on a true location, the average of the scrambled locations converges to the true location as the sequence gets longer.

Our solution is to de-randomize the scrambled locations. Specifically, when the Engine detects less than minimum movement by a user inside any private zone, it reuses the previous scrambled location rather than generating a new one. This way, attackers do not get new information even as the handset continues to send LBS queries, and therefore, the user's location privacy is protected.

5.6 Evaluation

We now evaluate LISA on G1 smartphones using real-world GPS traces of different movement patterns. First, we measure the energy consumption of a few benchmarks on a G1 phone. We then describe the traces collected for the evaluation, including walking, driving, riding a bus, and combinations thereof. Finally, we thoroughly analyze how LISA performs for the collected traces in terms of privacy protection and energy consumption.

5.6.1 Measurement of Energy Cost

To quantify the energy cost of LISA on the G1, we implemented a benchmark software on the Android platform, and used a digital multimeter to measure the electric current the phone drew when running one of the following benchmarks: baseline (the normal background activities such as radio beacons, the Android OS, and the screen), GSM receive, GSM send, Toggle GSM (turn on/off the GSM radio), GPS (get a location reading from the integrated GPS), and Kalman Filter. We did not consider the T-Mobile 3G network due to lack of coverage in our town. Because these benchmarks represent all constituent activities of LISA, their energy consumptions are used to estimate the energy consumption of our system.

Typically, a LBS query ascribes its energy consumption on a smartphone to the following parts: (1) getting the current location, (2) scrambling the location using the extended Kalman filters, (3)

sending the query to the proxy server, and (4) receiving the returned POI information (GSM). Such fine-grained measurements also enable us to evaluate the relative significance of each hardware component's energy cost and efficiently arrange different tasks to achieve the desired privacy level with minimum energy cost.

We set up experiments as follows. A 5V DC adapter was modified to power the G1, avoiding the variability of the battery's output voltage over time. The main battery was removed to avoid effects of charging battery during the experiments. We cut the power cord and connected an Agilent 34401A Digital Multimeter to measure the electric current drawn by the phone. The multimeter is capable of sampling system current at the rate of 10-20 samples per second. The multimeter was connected to a laptop which logged the time-stamped readings of instantaneous current. The G1 phone executes only one benchmark at a time for multiple times. For instance, the "GSM send" benchmark periodically transmits network packets of different sizes (ranging from 500 to 1M bytes) to a remote server through the T-Mobile GSM network. However, the measurement is not only the cost for GSM transmission but also for those background processes. To estimate the net energy cost, we measured the baseline energy consumption when the G1 executed a "null" task.

Experiment		Average Energy Consumption (Joules)
GSM Receive	500 bytes	3.05843
	1000 bytes	4.16094
	10000 bytes	5.67318
	50000 bytes	8.45077
	100000 bytes	12.4134
	500000 bytes	38.3875
	1000000 bytes	75.0637
GSM Send	50 bytes	4.66598
	200 bytes	4.67434
Toggle GSM		26.9085
GPS		2.1
Kalman Filter		0.00075
Baseline with screen off (per second)		0.028

Table 5.1: Energy consumption of different benchmarks on a T-Mobile G1 phone

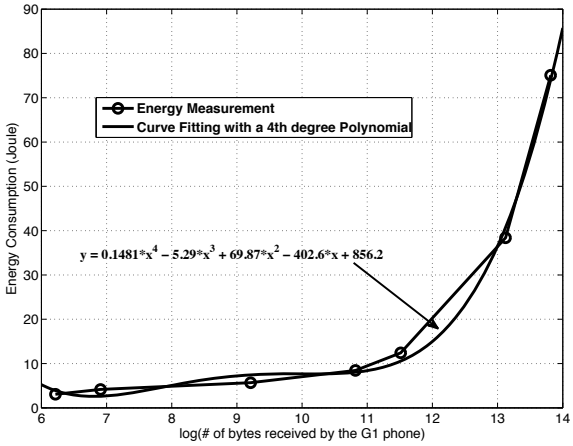


Figure 5.5: Average energy cost for receiving packets through the T-Mobile GSM network and curve-fitting them with a cubic polynomial.

The measurements of energy consumption are summarized in Table 5.6.1. Because the size of responses from the proxy can vary over a wide range, we curve-fit the energy consumptions of GSM receive on different sizes of packets, as shown in Figure 5.5.

5.6.2 Real-world Trace Collection

To collect real-world traces, we also implemented a GPS trace collector on the Android platform. G1 phones have a built-in GPS receiver, and the Android system provides APIs for querying GPS records in real time. Hence, the collector can obtain updates of the mobile phone's current location every 1 or 2 seconds. Note that in a real application, a location is only needed when the mobile is requesting some LBS services in order to conserve the battery power. In our experiments, we record the GPS readings at a much higher frequency so that we can mimic different user request patterns. For example, we model the arrival time of LBS queries with Poisson distributions with different mean intervals.

We carried the G1 phone on three daily routines: walking between home and office, taking a campus bus (that includes both walking and driving), driving around the town. The traces are shown in Figures 5.7, 5.9 and 5.11, where each red triangle represents a POI. These three patterns can represent user movements under a wide range of application scenarios.

Figure 5.6: Each POI is shown as a pushpin on the map

Because our traces were collected on campus, we obtained a geo-database of over 500 campus buildings, and added all campus bus stops into the database. Figure 5.6 shows the POIs in a sub-area of our campus.

5.6.3 Simulation Setup

In this subsection, we evaluate the performance of LISA via Matlab-based simulation. Although we have implemented LISA on the G1 phone, we decided to use simulation instead of field tests because (1) performing field tests are time-consuming and the results are difficult to reproduce; (2) the simulation allows us to easily adjust a variety of parameters (e.g., inter-arrival time between successive location updates) to create different scenarios and repeat each scenario multiple times; and (3) the simulation results should match those of field tests, because we have the same implementation in Matlab as in the G1 phone, use the real-world traces and the POI database to drive the simulation, and estimate the energy consumption using benchmark measurements on the G1.

The system parameters used in the simulation are listed in Table 5.2 and detailed below.

- To simulate the usage of LBS services, we assume that the number of LBS queries a mo-

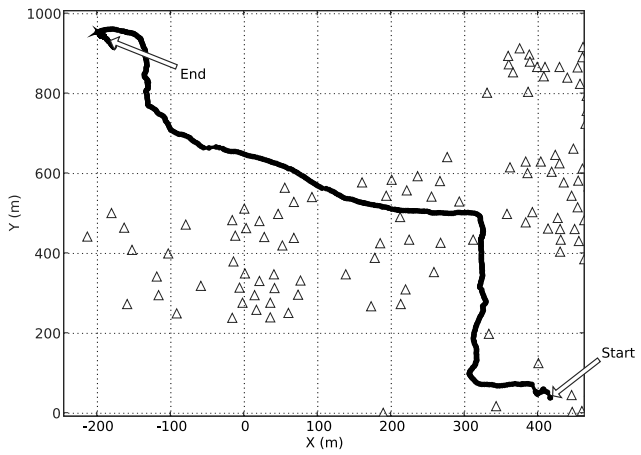


Figure 5.7: Traces of walking between home and office

Parameter	Default value
mean inter-arrival time of LBS queries (λ)	60 seconds
look-ahead window size (t_0)	0.5λ
Total number of POI	590
Anonymity Requirement	10-unobservability
Entropy threshold	3.321928
LBS query range (L)	400m
Range of observation deviation (r)	[10m, 1000m]

Table 5.2: Default values for simulation parameters

bile user initiates is a Poisson process with the default mean inter-arrival time ($1/\lambda$) equal to 60 seconds. Then, the query arrival pattern can be generated as follows. First, the simulation engine loads a GPS trace file containing a sequence of GPS locations (p_i) and their corresponding timestamps T_i^P. Next, the engine generates a series of time points t_j where $t = t_j - t_{j-1}$ follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$. Last, the first GPS location p_i with $T_i^P \geq t_j$ is selected as the location where the mobile user submits the j-th LBS query.

- The look-ahead window determines how far in future the movement prediction goes. We set it to be proportional to the mean inter-arrival time.

- The local POI database has 590 entries, covering a region of about $70km^2$ and the average

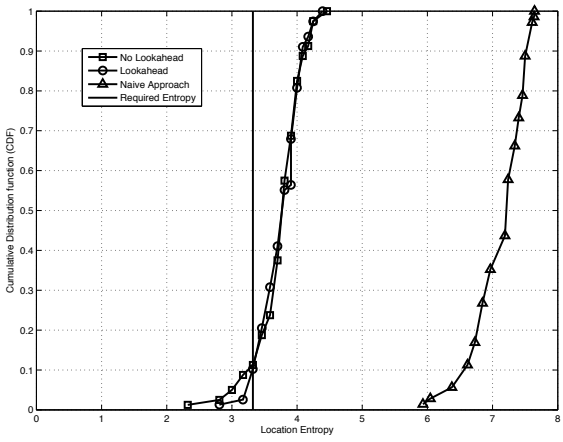


Figure 5.8: Location entropies of walking traces

POI density is 8.4 per km^2.

- A mobile user needs to scramble his location inside pre-defined privacy zones. Each zone has a privacy requirement based on the user's preference. The privacy requirement is defined in terms of the m-unobservability or the entropy $log(m)$ (see Section 5.3). In the simulation, the default anonymity requirement is set to *10-unobservability* that corresponds to an entropy value of 3.321928. This means that attackers can relate the mobile user's locations to at least 10 POIs.

- Each LBS query has a range, $L = 400m$. That is, the query region is a rectangle $(x-L, y-L, x+L, y+L)$ if the user's location (x,y) is not scrambled, or a rectangle $(x\prime - 2\delta_x - L, y\prime - 2\delta_y - L, x\prime + 2\delta_y + L, y\prime + 2\delta_y + L)$ if the location is scrambled as $(x\prime, y\prime)$ and the standard deviation of location estimations along X-axis and Y-axis are δ_x and δ_y, respectively.

- In our simulation, the default limit of r (Section 5.4 S6) between 10m and 1km.

In the rest of this section, we evaluate the effectiveness and efficiency of LISA. We also evaluate the impact of the above parameters on LISA's capability in scrambling the location information. Unless otherwise stated, the system parameters are set to their default values.

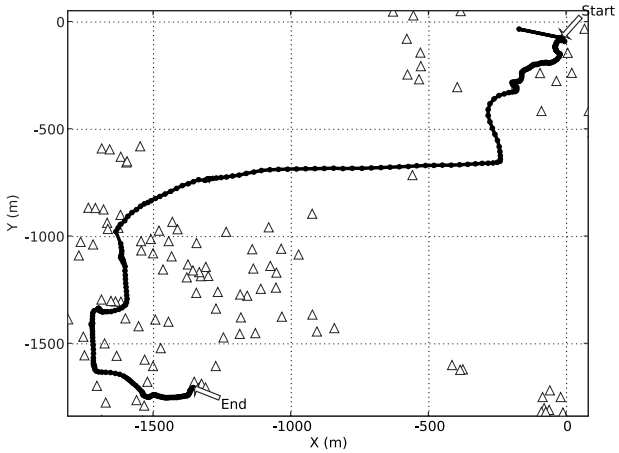


Figure 5.9: Traces of taking a campus bus

5.6.4 Performance Metrics

The main metric for privacy protection is the average location privacy entropy along each movement track. We need to look at an entire track because the location privacy at t_j depends on the privacy at t_{j-1}. Additionally, because the goal of LISA is to keep the privacy entropies above the requirements, it is important to evaluate the percentage of times the requirement can be met. Hence, we define the protection success rate as the percentage of LBS queries sent from private zones that satisfy the privacy requirements.

The direct measure for energy efficiency is energy consumption in Joules. We consider the energy consumption of the entire system, including the Kalman filter computation, GPS readings, and communication over the GSM network. To compare the communication costs, we also compare the size of the LBS query rectangles and the number of POIs returned to account for uneven distribution of POIs.

5.6.5 Evaluation Results

We compare the performance of LISA and its optimization (LISA-lookahead) with a naive approach (Naive), which always uses the maximum r to maximize unobservability.

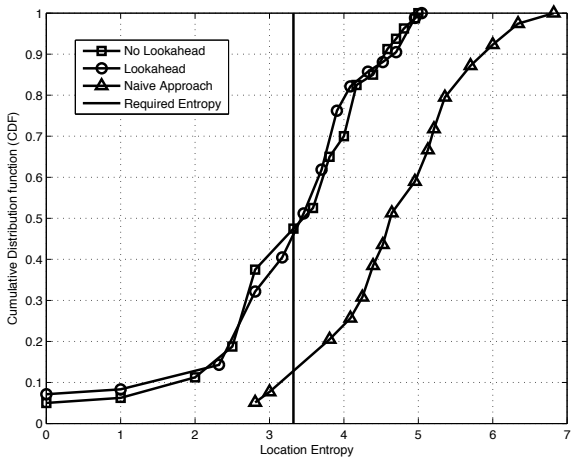


Figure 5.10: Location entropies of bus traces

Privacy Protection

In the experiment, we randomly pick a privacy zone with the default privacy requirement, and simulate LBS queries using a Poisson distribution. When the mobile user enters the private zone, LISA scrambles the location information to satisfy the privacy requirement. To evaluate the effectiveness of LISA, we simulate a location privacy attack against the mobile user as follows. Attackers are assumed to have gained access to all the history LBS queries from the mobile user. Their goal is to estimate the current location of the mobile user inside the private zone from a sequence of the user's history locations (those sent in the private zone are scrambled, and rest are not), and thereby infer the POIs to which the mobile user is related. The results of attacks are quantified by the location entropy computed from Eq. (5.3), i.e., the smaller the location entropy, the less location privacy is protected. For each LBS query issued within the private zone, a success is counted if the location entropy of attackers' estimation is above the privacy requirement predefined by the user (Table 5.2). We repeat the experiments 50 times for each of 3 trace types and compute the *success rates* as the number of successes over the total number of LBS queries.

Table 5.3 compares the success rates of different traces across three approaches (LISA, LISA-lookahead and naive). We also collect all the location entropy values and plot their CDFs (cumulative distribution functions) in Figures 5.8, 5.10 and 5.12. The average location entropies for different traces are also summarized in Table 5.4.

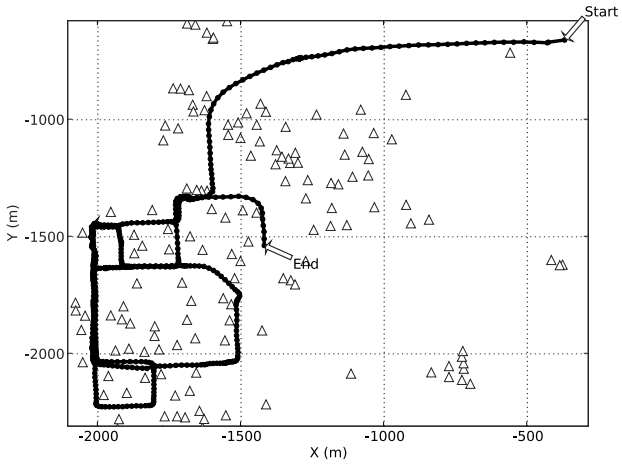


Figure 5.11: Traces of driving around the town

From Tables 5.3 and 5.4, we can make the following observations. First, LISA and its look-ahead optimization can satisfy the privacy requirement by 70-90%. This is also demonstrated by the average entropy values in Table 5.4 which all exceed the user requirement, 3.32. Second, the look-ahead optimization provides, on average, 5% improvements in success rate over the original LISA for walking and campus bus traces. However, very little improvement is made for driving traces. This is because the look-ahead optimization relies on accurate prediction of the user' entrance into the private zone so that it can adjust the location uncertainty beforehand. In the driving traces (Figure 5.11), the car took many sharp turns and moved faster than pedestrians or a campus bus, so location prediction becomes very inaccurate and the optimization makes no improvement. Third, the naive approach which adopts the maximum query range provides the highest protection of location privacy, as indicated by near-perfect success ratios. However, there is a trade-off between the communication cost and the energy consumption incurred by the naive approach as we will show later.

	Success Rate		
	LISA	LISA-Lookahead	Naive
Walking	0.92	0.973333	1
Campus Bus	0.683333	0.726667	0.99
Driving	0.795294	0.796471	0.998824

Table 5.3: Success rates for different types of traces using different algorithms

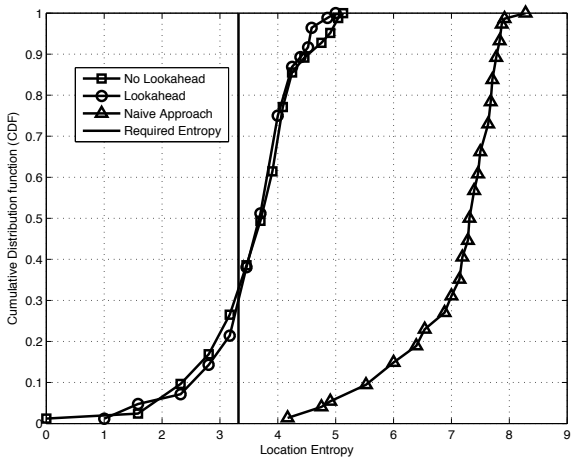


Figure 5.12: Location entropies of driving traces

	Average Entropy		
	LISA	LISA-Lookahead	Naive
Walking	3.774122	3.800483	7.095277
Campus Bus	3.165494	3.346328	4.705274
Driving	3.637652	3.618186	6.993945

Table 5.4: Success rates for different types of traces using different algorithms

Figure 5.13 shows the success rate of the three approaches while varying the query interval. The success rate is shown (on y-axis) for different mean intervals of LBS queries. As the interval increases, both A_k and G_k increase, the prediction covariance $P_{k+1|k}$ increases significantly, meaning that the location predictions becomes more inaccurate as indicate by Eq. (5.7). Therefore, it becomes very difficult for the lookahead optimization to correctly predict if the user enters a private zone at the next step, and thus, loses its advantage and even gets worse due to incorrect prediction, as the interval increases.

As shown in Figure 5.14, the overall average entropy decreases as the interval increases. This is because, with the POIs clustered rather than evenly distributed, it becomes more likely that the user enters a private zone at the two ends of the track (walking) where POIs are sparse, as the interval increases.

Figure 5.15 shows how the success rate changes as the maximum observation deviation, r,

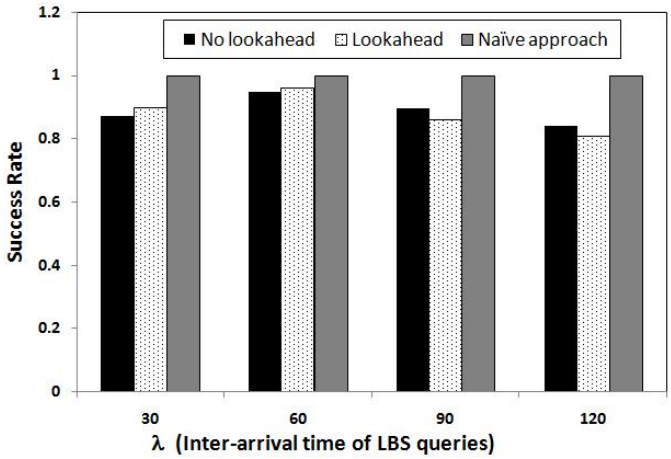


Figure 5.13: Success rates for different λ

increases. When r is small and the variance of the location estimation is bounded by r, the success rate is limited. As the deviation bound grows, the variance of location estimation also increases, allowing larger uncertainty regions to satisfy the privacy requirement. However, as the deviation bound continues to grow, even larger uncertainty regions do not improve the success rate. This is because we have a limited set of POIs, and the expanded areas do not cover more POIs.

Figure 5.16 depicts the relationship between the success rate and the look-ahead window size. The look-ahead window size determines how far in the future LISA predict a mobile user's movement. At a small window size, LISA estimates the user's location only in the near future (e.g., a few seconds into the future) and may fail to predict the situation when the user enters the private zone. In such cases, the look-ahead optimization provides no additional benefits. As the window size grows close to the mean interval between two consecutive LBS queries from the user, LISA's prediction accuracy also increases. Because success prediction allows LISA to proactively adjust the location uncertainty, an appropriate window size will improve LISA's success rate. However, as the window size continues to increase, LISA tends to over-predict the distance that the mobile user may travel before s/he sends the next LBS update. This often degrades the prediction accuracy and negatively affects the success rate, for example, when the window size equals 1λ or 1.25λ in Figure 5.16

Finally, we evaluate the effectiveness of the de-randomization optimization by comparing the LISA's success rate when the optimization is enabled, with the success rate when it is disabled.

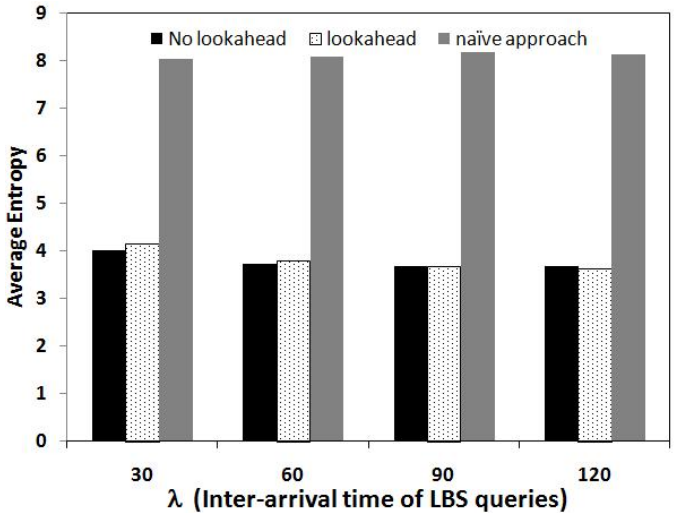


Figure 5.14: Location entropies for different λ

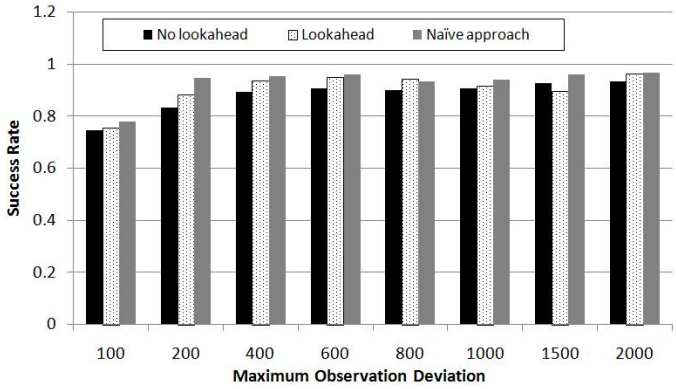


Figure 5.15: Success rates for different *r* values (maximum observation deviation)

We simulate the scenario where a mobile user stays at the same location inside the private zone for a long time period during which s/he issued 40 LBS queries out of 70 in total. LISA without the de-randomization optimization handles the queries in normal way where, for each query, LISA manipulates the location uncertainty by generating a random location offset. On the other hand, LISA with the optimization sends the same location information as the last LBS queries as long as the mobile user stays at the same position. The experimental results are summarized in Table 5.5, showing that the de-randomization optimization significantly improves the success rate. This is because by sending the same location updates, a mobile user does not leak additional information to the any unauthorized party, and thus stands a better chance to successfully protect his/her location

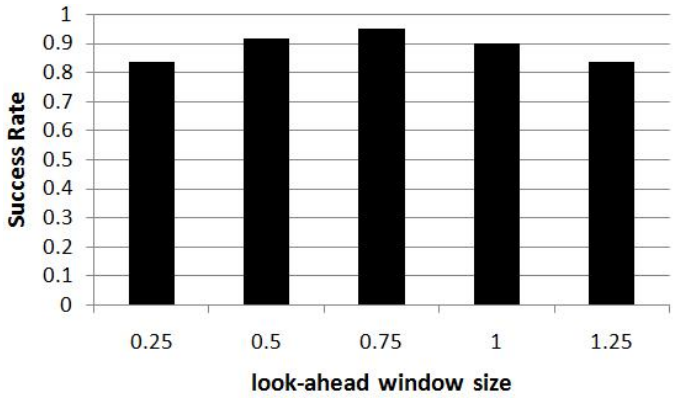


Figure 5.16: Success rates for different look-ahead window sizes

	No De-randomization			De-randomization
	No Lookahead	Lookahead	Naive	
Success Rate	0.769	0.772	0.937	0.917
Average Entropy	3.543	3.543	6.832	5.660

Table 5.5: Effectiveness of de-randomization optimization

privacy.

Energy-Efficiency

In this subsection, we comparatively evaluate LISA's energy-efficiency, using three metrics: (1) expansion of the query region (km^2), (2) additional POIs in the query responses, and (3) additional energy consumption. LISA expands the region in the location queries to scramble the mobile user's true location. However, a larger query region implies a higher energy cost because more POIs may be found in a larger query region, and sending them back to the mobile user incurs a higher communication overhead. LISA tries to balance the privacy protection and energy consumption by searching for a minimum possible query region that satisfies the m-unobservability so that the extra energy cost can be minimized. Figure 5.17 plots the CDF of additional query regions and Table 5.6 presents the average values. One can see that, even though the naive approach provides the strongest privacy protection, it leads to the query region that are 10 times larger than LISA. As

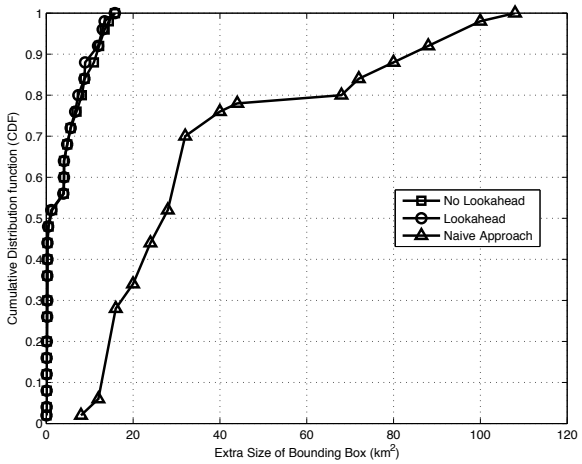


Figure 5.17: CDF of expanded query range

a result, about 5 times more POIs[2] are transferred back to the user. In addition, it is also shown that LISA-lookahead receives the smallest number of POIs, due mainly to the prediction mechanism in lookahead optimization. Finally, we compute the additional energy consumption incurred by all three approaches with the measured benchmark values. The average additional energy cost and the CDF functions are summarized in Table 5.6 and plotted in Figure 5.18, showing that LISA incurs less than 100 joule additional energy consumption for each track, and the look-ahead optimization reduces the additional energy cost by 37% over the non-lookahead LISA and 52% over the naive approach. As a result, LISA can achieve strong location privacy protection while consuming a reasonable amount of energy.

	LISA	LISA-Lookahead	Naive
average number of extra POIs	1128	735	1416
average area of the extra query range (km^2)	3.95	3.48	38.1
average energy cost (Joule)	83	52	108

Table 5.6: Communication and energy cost of three approaches

[2]The number in Table 5.6 is the total number of POIs returned along an entire trace rather than in a single LBS query response

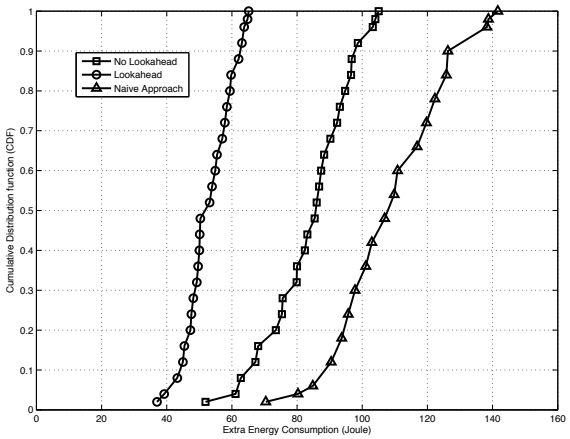


Figure 5.18: CDF of additional energy cost

5.7 Related Works

In this section we discuss prior work on location privacy in three aspects: threats, metrics, and protection.

5.7.1 Location privacy threat

Prior work has raised two types attacks: location disclosure, and movement tracking. In the former, Hoh *et al*. [101] and Krumm [111] show that a driver's home location can be inferred from GPS data collected on his vehicle even though the location data is pseudonymized. Moreover, Matsuo [112] uses indoor location information of a user to infer a variety of personal information, such as work role, smoker, coffee drinker, and even age. In the latter, Gruteser and Hoh show that coherent, individual tracks can be reassembled from completely anonymized GPS data from three or even five users [102, 103] by using Multiple-Hypothesis Tracking.

5.7.2 Location privacy metrics

There is not yet a standard for quantifying location privacy, but most of the location privacy metrics are uncertainty-based. That is, a user's location privacy is protected better if attackers are more unsure about differentiating the user from others within an anonymity set [25, 24, 106], linking two pseudonyms of the person uses outside a mix zone or in a wireless LAN [26, 113], or

distinguishing certain POIs related to the user in our approach. Hence, the level of location privacy is determined by the size of the anonymity set, the number of users in the mix zone, or the number of POIs that can be related to the user's location.

Entropy is just another form of representing uncertainty. Jiang [113], Hoh [106] and we adopt entropy, as it is a more convenient form in the individual application scenarios.

Aside from uncertainty-based metrics, Hoh *et al*. [103] also used the expected error between the attackers' estimation and a user's true location to measure location privacy.

5.7.3 Location privacy protection

There have been many computational measures proposed and tested for protecting location privacy, which are based on anonymity, unlinkability, and unobservability, according to the terminology proposed by Pfitzmann and Koehntopp [107].

In the first class, location privacy is quantified as the uncertainty of user identity. By spatially [25] or temporally [106] cloaking a user's location information, or both [24], the user cannot be distinguished from at least $k-1$ other users. The higher k is, the more the user's location privacy is protected.

The second class unlinks the two pseudonyms of a user, so attackers cannot accumulate enough history on the user to infer any personal information. Beresford and Stajano [26] proposed a mix zone in which a number of users change to new, unused pseudo names, so that attackers cannot link people going into the mix zone with those coming out of it. Jiang and Wang [113] unlink different pseudonyms of the same user with silent periods between different pseudonyms, which is planned using well-known user mobility patterns.

In the third class, our approach scrambles a user's locations such that the estimated location is *m-unobservable*, i.e., it can be related to at least m points of interests. Hence, attackers cannot relate any particular POI to the estimated location, and thus, their ability to infer the user's private information is significantly weakened.

LISA introduces a new dimension of uncertainty, and can be combined with the approaches in the other two classes to enhance privacy protection.

Additionally, LISA does not rely on a trustworthy third party server like in [25, 24, 26]. This limits privacy attacks to individual cell phones, thus significantly reducing the impact of attacks,

and lowers the trust requirements from mobile users, thus simplifying the implementation and deployment of LBSs.

Our work is motivated by Hoh and Gruteser [103], who studied the movement tracking attack that exploits multiple hypothesis tracking (MHT) to identify completely anonymized GPS traces from multiple users. Their solution was perturbing the original locations by the radius R, such that two original paths that are approximately parallel become intersected. Therefore, attackers are confused at which user takes on which path. This is similar to Meyerowitz and Choudhury's [114] creating confusion at crossroads.

5.8 Conclusion

We have proposed a new approach, called the *Location Information ScrAmbler* (LISA), to protecting the location privacy of mobile users. The key idea is *m-unobservability*, which disables the distinguishability of the POIs near each user, and therefore, weakens the attackers' capability of inferring the user's private information or mobility patterns from his locations. Based on a simple mobility model, LISA achieves *m-unobservability* by adding measurement noise to the location information provided to the proxy server or LBS providers.

LISA introduces a new, orthogonal dimension of uncertainty, and can be combined with previous approaches to enhance location privacy. Moreover, it eliminates the need for any trustworthy server, thus lowering the risk of attacks and simplifying the implementation and deployment of LBS systems.

We evaluated the performance of LISA using both experiments on T-Mobile G1 and Matlab simulations. The results show that LISA can effectively protect location privacy with good energy efficiency.

CHAPTER 6

Conclusions and Future Work

This dissertation has focused on how to build energy-efficient information collection and dissemination systems in wireless sensor networks. We first summarize our main contributions and then discuss the future work.

6.0.1 Main Contributions

This dissertation makes the following contributions.

- In-network data aggregation. We propose a new approach, called the *Opportunistic Data Aggregation* (OPAG), to *in-network* data aggregation with zero computation error and good tolerance to moderate message losses in wireless sensor networks. By multiplexing message space, OPAG opportunistically exploits multi-path routing which is more energy-efficient than the usual scheme based on retransmissions. This is motivated by a key observation that, when sending a message, the radio may consume much more energy for idle listening during the backoff period and the time to wait for its acknowledgment than transmitting the data bits. We implemented OPAG on TinyOS-2.x and the TMote Sky node, and evaluated its performance on the Motelab Testbed, and TOSSIM. Experimental and simulation results show that OPAG consumes much less energy than TAG and Sketch/SD while incurring similar aggregation error under relatively good network connectivity, and the data selection scheme is effective when contention for the message space is moderate.

- Post-deployment debugging. We propose a data-centric approach called *post-deployment performance debugging* (PD2). PD2 focuses on the data flows that an application generates,

and relates poor application performance to significant data losses, energy consumptions, or latencies of some data flows (problematic data flows) as they go through the software modules on individual nodes and through the network. PD2 derives a few inference rules based on the data dependencies between different software modules, as well as between different nodes, and use them to trace back in each problematic flow. Then, PD2 turns on the performance monitoring of, and collects debugging information from, only those modules and nodes that the problematic flows go through. Finally, PD2 provides the debugging information to help users isolate the causes of poor performance. Our approach can be applied to various sensor networks applications, such as data collection/aggregation and event-driven applications with mobile data sinks. We have implemented PD2 on TinyOS and evaluated it on a real WSN testbed. Our experimental results show that PD2 can help users quickly locate the possible sources of problems, such as code bugs, weak links, and radio interference. Depending on the hop-count distance between the source of the problem and the data sink, PD2's energy consumption (communication overhead) is shown to be only 5–10% of that of collecting debugging information from all nodes.

- We propose a distributed location service protocol (DLSP) for hybrid networks. Through a rigorous analysis of DLSP, we derive the condition for achieving a high packet-delivery ratio, and show how to configure the protocol parameters to ensure the scalability of DLSP. DLSP is found to be scalable if the mobile's speed is below a certain fraction of the packet-transmission speed which depends on the underlying movement threshold. For example, if the movement threshold for the location servers at the lowest level in the hierarchy is set to the radio range, the mobile's speed limit is one-tenth of the packet-transmission speed. The mobile's theoretical speed limit is one-fifth of the packet-transmission speed, beyond which DLSP cannot scale regardless of the movement threshold. These analysis results are confirmed by extensive simulation. We also introduce an optimization technique that significantly reduces the protocol overhead and a greedy adaptation mechanism that improves overall energy-efficiency. We evaluate DLSP with extensive simulation using two MAC protocols and three mobility models. The evaluation results confirm our analysis. We also proposed an optimization technique, DLSP-SN that can reduce the location-update overhead by 70% or more, while its query-delivery ratio is even better than DLSP when the

mobile's speed is high. In order to make a tradeoff between update and data-delivery costs, we present a greedy adaptation mechanism, DLSP-ASN, which can significantly improve overall energy-efficiency.

- As use of location-based services (LBSs) is becoming increasingly prevalent, mobile users are more and more enticed to reveal their location information to untrustworthy third parties, making it very difficult to protect their location privacy. To cope with this problem, we propose a new privacy protection approach based on *m-unobservability*, called the *Location Information ScrAmbler* (LISA) that prevents the distinguishability of points of interest (POIs) to a mobile user, thereby weakening attackers' ability to infer the user's private information or mobility patterns. LISA adjusts the location noise level in location service queries and ensures that the uncertainty of the attackers' location estimation satisfies *m-unobservability*. By protecting location privacy locally on each mobile user's devices, LISA eliminates the need for the trustworthy third-party servers in previous approaches, thus facilitating deployment of LBSs. However, this also incurs extra energy consumption on mobile devices, so LISA explores the tradeoff between the estimation uncertainty and the energy consumption to achieve both strong privacy preservation and efficient energy conservation. Our extensive evaluation using real-world traces of human and vehicle mobility patterns demonstrates the efficacy and efficiency of LISA.

6.0.2 Future Work

Our work on energy-efficient information collection and dissemination in wireless sensor networks can be extended in the following aspects.

- In-network data aggregation. In Chapter 2, OPAG only handles simple aggregate queries, such as sum, average, and count. Many sensor network applications involve complex queries, such as histogram and sketch. We will investigate the tradeoff between the accuracy of the intermediate results and the degree of tolerance to message loss by space-multiplexing messages for these complicate queries.

- Post-deployment debugging. In Chapter 3, we proposed PD2 for debugging data collection applications. We will enhance PD2 to accommodate applications other than data collection,

especially event-driven applications. PD2 can be extended to "record and replay" the data flow triggered by an event and then treat it as a special data-collection application.

- Location service protocol. In Chapter 4, we presented a distributed location service protocol (DLSP) which let sensor nodes query the location of a mobile sink for every data message. In future we would like to study how location caching can affect the performance and energy efficiency of the location service.

- Location privacy protection. In Chapter 5, LISA exploits a simple mobility model. We will explore other mobility models, and let LISA adaptively choose appropriate models to improve its overall performance.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, 2002.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks (Elsevier*, vol. 46, pp. 605–634, 2004.

[3] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *Sensys*, 2004.

[4] V. Narasimhan and K. Bever, "Greenhouse asset management using wireless sensor-actor networks," in *UBICOMM*, 2007.

[5] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Sensys*, 2008.

[6] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden, "Cartel: A distributed mobile sensor computing system," in *Sensys*, 2006.

[7] hravan Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: Sharing and querying content through mobile phones and social participation," in *Mobisys*, 2008.

[8] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. Eisenman, X. Zheng, and A. Campbell, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *Sensys*, 2008.

[9] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," in *OSDI*, December 2002.

[10] J. Considine, G. K. F. Li, and J. Byers, "Approximate aggregation techniques for sensor databases," in *ICDE*, 2004.

[11] S. Nath, "Synopsis diffusion for robust aggregation in sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, no. 4, 2004.

[12] A. Manjhi, S. Nath, and P. B. Gibbons, "Tributaries and deltas: Efficient and robust aggregation in sensor network streams," in *SIGMOD*, 2005.

[13] A. Campbell, S. Eisenman, K. Fodor, N. Lane, H. Lu, E. Miluzzo, M. Musolesi, R. Peterson, and X. Zheng, "Transforming the social networking experience with sensing presence from mobile phones (demo abstract)," in *Sensys*, 2008.

[14] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *SenSys*, 2005.

[15] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing," in *SenSys*, 2003.

[16] G. Tolle and D. Culler, "Design of an applicationcooperative management system for wireless sensor networks," in *In Proc. of the 2nd European Workshop on Wireless Sensor Networks (EWSN*, 2005.

[17] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, 2000, pp. 243–254.

[18] S. Lee, B. Bhattacharjee, and S. Banerjee, "Efficient geographic routing in multihop wireless networks," in *MobiHoc '05*, 2005.

[19] P. F. Tsuchiya, "The landmark hierarchy: A new hierarchy for routing in very large networks," in *Proceedings of SIGCOMM'88*. ACM, Aug. 1988.

[20] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica, "Beacon vector routing: Scalable point-to-point routing in wireless sensornets," in *NSDI 2005*, 2005.

[21] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *MobiCom '00*, 2000.

[22] Y. Xue, B. Li, and K. Nahrstedt, "A scalable location management scheme in mobile ad-hoc networks," in *LCN '01*, 2001.

[23] S. M. Das, H. Pucha, and Y. C. Hu, "Performance comparison of scalable location services for geographic ad hoc routing," in *INFOCOM '05*, 2005.

[24] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," in *ICDCS*, 2005.

[25] M. Gruteser, D. Grunwalddepartment, and C. Science, "Anonymous usage of location-based services through spatial and temporal cloaking," in *In MobiSys*, 2003, pp. 31–42.

[26] A. R. Beresford and F. Stajano, "Location privacy in pervasive computing," in *Pervasive Computing*, 2003.

[27] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acqusitional query processor for sensor networks," in *SIGMOD*, June 2003.

[28] Y. Yao and J. E. Gehrke, "Query processing in sensor networks," in *First Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2003.

[29] P. F. jolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Computer and System Sciences*, 1985.

[30] N. Reijers, G. Halkes, and K. Langendoen, "Link layer measurements in sensor networks," in *Int. Conf. on Mobile Ad-hoc and Sensor Systems*, 2004.

[31] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *SIGCOMM*, 2004.

[32] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: A wireless sensor network testbed," in *IPSN'05, Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, 2005.

[33] "Moteiv," http://www.moteiv.com.

[34] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04*, 2004.

[35] W. Ye and J. Heidemann, "Ultra-low duty cycle mac with scheduled channel polling," in *Conference On Embedded Networked Sensor Systems*, 2006.

[36] N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: ultra-low power data gathering in sensor networks." in *IPSN*, 2007.

[37] "The collection tree protocol, tinyos-2.x tep 123," http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html.

[38] R. K. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher, "Datalink streaming in wireless sensor networks," in *Proceedings of ACM SenSys*, 2006.

[39] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Toward sophisticated sensing with queries," in *IPSN*, 2003.

[40] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann, "An evaluation of multi-resolution storage for sensor networks," in *ACM SenSys*, 2003.

[41] P. Levis and N. Lee, "Tossim: Accurate and scalable simulation of entire tinyos applications," in *SenSys*, 2003.

[42] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *IEEE SPNA*, 2003.

[43] F. Ye, G. Zhong, S. Lu, and L. Zhang, "A robust data delivery protocol for large scale sensor networks," in *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, 2003.

[44] A. Boulis, S. Ganeriwal, and M. Srivastava, "Aggregation in sensor networks: an energy-accuracy trade-off," in *SNPA*, 2003.

[45] S. Dulman, T. Nieberg, J. Wu, and H. Havinga, "Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks," in *Wireless Communications and Networking Conference*, 2003.

[46] C. L. Barrett, S. J. Eidenbenz, L. Kroc, M. Marathe, and J. P. Smith, "Parametric probabilistic sensor network routing," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 2003, pp. 122–131.

[47] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, 2000.

[48] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *Mobile Computing and Communications Review*, vol. 1, no. 2, 2002.

[49] "Tmote sky datasheet, moteiv," http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf.

[50] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *ACM SIGPLAN*, 2003.

[51] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys*. ACM Press, 2004, pp. 188–200.

[52] J. jin Lim and K. Shin, "Gradient-ascending routing via footprints in wireless sensor networks," in *RTSS*, 2005.

[53] "Intel imote data sheet," http://support.intel.com/research/downloads/imote-ds-101.pdf.

[54] Z. Chen and K. Shin, "Opag: Opportunistic data aggregation in wireless sensor networks," in *RTSS*, 2008.

[55] Z. Chen, M. Cho, and K. Shin, "Distributed location service protocol for a hybrid network of static sensors and mobile actors," in *RTSS*, 2008.

[56] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *Proceedings of the 8th annual international conference on Mobile computing and networking*, 2002, pp. 148–159.

[57] B. Titzer, D. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," in *In Proc. of the 4th Intl. Conf. on Information Processing in Sensor Networks (IPSN*, 2005, pp. 477–482.

[58] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "Emstar: a software environment for developing and deploying wireless sensor networks," in *In Proceedings of the 2004 USENIX Technical Conference*, 2004, pp. 283–296.

[59] Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, and L. Luo, "Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks," in *Sensys*, 2008.

[60] M. Khan, H. Le, H. Ahmadi, T. Abdelzaher, and J. Han, "Dustminer: troubleshooting interactive complexity bugs in sensor networks," in *Sensys*, 2008.

[61] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong, "Passive diagnosis for wireless sensor networks," in *Sensys*, 2008.

[62] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," in *SOSP*, 2003.

[63] M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis, "Visibility: a new metric for protocol design," in *SenSys*, 2007.

[64] W. Archer, P. Levis, and J. Regehr, "Interface contracts for tinyos," in *In Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN*. ACM Press, 2007, pp. 158–165.

[65] K. Romer and M. Ringwald, "Increasing the visibility of sensor networks with passive distributed assertions," in *REALWSN*, 2008.

[66] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, Aug. 2008.

[67] S. Olariu, M. Eltoweissy, and M. Younis, "ANSWER: autonomous wireless sensor network," in *Q2SWinet '05*, 2005.

[68] Č. Westpha, "Little tom thumb went straight home, asymptotic behavior of a routing protocol in ad hoc networks with a mobile access point," in *INFOCOMM*, 2007.

[69] J. Ortiz, C. R. Baker, D. Moon, R. Fonseca, and I. Stoica, "Beacon location service: a location service for point-to-point routing in wireless sensor networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 166–175.

[70] W. Kieß, H. Füßler, J. Widmer, and M. Mauve, "Hierarchical location service for mobile ad-hoc networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, no. 4, 2004.

[71] R. Flury and R. Wattenhofer, "MLS: an efficient location service for mobile ad hoc networks," in *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, 2006, pp. 226–237.

[72] I. Stojmenovic, "A routing strategy and quorum based location update scheme for ad hoc wireless networks," University of Ottawa, Tech. Rep., 1999.

[73] A. C. Viana, M. D. de Amorim, S. Fdida, Y. Viniotis, and J. F. de Rezende, "Easily-managed and topology-independent location service for self-organizing networks," in *MobiHoc '05*, 2005.

[74] S. Sivavakeesar and G. Pavlou, "Scalable location services for hierarchically organized mobile ad hoc networks," in *MobiHoc '05*, 2005.

[75] W. Ye and J. Heidemann, "Ultra-low duty cycle MAC with scheduled channel polling," USC/Information Sciences Institute, Tech. Rep. ISI-TR-2005-604b, 2005.

[76] Y. Li, W. Ye, and J. Heidemann, "Energy and latency control in low duty cycle MAC protocols," in *WCNC '05*, 2005. [Online]. Available: http://www.isi.edu/ johnh/PAPERS/Li05a.html

[77] T. Goff, N. B. Abu-Ghazaleh, D. S. Phatak, and R. Kahvecioglu, "Preemptive routing in ad hoc networks," in *MobiCom '01*, 2001.

[78] H. Kim and J. Hou, "Improving protocol capacity with model-based frame scheduling in IEEE 802.11operated WLANs," in *MobiCom '03*, 2003.

[79] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *INFOCOM '02*, 2002.

[80] T. Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *SenSys '03*, 2003.

[81] A. El-Hoiydi and J. Decotignie, "WiseMAC, an ultra low power mac protocol for the WiseNET wireless sensor network," in *ISCC '04*, 2004.

[82] A. Savvides, C.-C. Han, and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *MobiCom '01*, 2001.

[83] L. Hu and D. Evans, "Localization for mobile sensor networks," in *MobiCom '04*, 2004.

[84] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage," in *WSNA '02*, 2002.

[85] "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/.

[86] "MicaZ," http://www.xbow.com/products/productsdetails.aspx?sid=101.

[87] J. Yoon, M. Liu, and B. Noble, "Sound mobility models," in *Proceedings of ACM/IEEE MobiCom 2003*, Sept. 2003.

[88] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communication and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking*, 2002.

[89] S. Singh and C. S. Raghavendra, "PAMAS – power aware multi-access protocol with signalling for ad hoc networks," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 3, pp. 5–26, 1998.

[90] Q. Zheng, X. Hong, and S. Ray, "Recent advances in mobility modeling for mobile ad hoc network research," in *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*. New York, NY, USA: ACM, 2004, pp. 70–75.

[91] N. Sadagopan, F. Bai, B. Krishnamachari, and A. Helmy, "Paths: analysis of path duration statistics and their impact on reactive manet routing protocols," in *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. New York, NY, USA: ACM, 2003, pp. 245–256.

[92] A. SHARMA and J. E. VASCELLARO, "Phones will soon tell where you are," http://online.wsj.com/article/SB120666235472370235.html, March 2008.

[93] The Economist Magazine, Special Report, April, 2008, "Location, location, location it matters."

[94] Mobile Internet Capital Inc, "Our investment targets," http://www.mickk.com/en/eclassify1.html.

[95] Yahoo! Inc., "Fireeagle," http://fireeagle.yahoo.net, Nov 2008.

[96] Six Apart, "Mobile type," http://www.movabletype.com, Nov 2008.

[97] NextBus Inc., "Nextbus," http://www.nextbus.com.

[98] Citysense Inc., "Citysense," http://www.citysense.com.

[99] Computer Science and Telecommunications Board, "It roadmap to a geospatial future," Nov 2003.

[100] J. Krumm, "Inference attacks on location tracks," in *In Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive), volume 4480 of LNCS*. Springer-Verlag, 2007, pp. 127–143.

[101] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Enhancing security and privacy in traffic-monitoring systems," in *Pervasive Computing*, 2006.

[102] M. Gruteser and B. Hoh, "On the anonymity of periodic location samples," in *Security in Pervasive Computing*, 2005.

[103] B.Hoh and M. Gruteser, "Protecting location privacy through path confusion," in *SecureComm*, 2005.

[104] K. John, "A survey of computational location privacy," *Personal and Ubiquitous Computing*, 2008.

[105] T. K. Moon and W. C. Striling, *Mathematical Methods and Algorithms for Signal Processing*. Prentice-Hall, 2000.

[106] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J. C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson, "Virtual trip lines for distributed privacy-preserving traffic monitoring," in *MobiSys*, 2008.

[107] A. Pfitzmann and M. Hansen, "Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management - a consolidated proposal for terminology," http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, Feb. 2008, v0.31.

[108] C. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[109] X. R. Li and V. P. Jilkov, "Survey of maneuvering target tracking. part i. dynamic models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 1333– 1364, 2004.

[110] I. Constandanche, M. Sayler, S. Gaonkar, R. R. Choudhury, and L. Cox, "Enloc: Energy efficient localization for mobile phones," in *Infocom*, 2009.

[111] J. Krumm, "Inference attacks on location tracks," in *Pervasive Computing*, 2007.

[112] Y. Matsuo, N. Okazaki, K. Izumi, Y. Nakamura, T. Nishimura, and K. Hasida, "Inferring long-term user properties based on users location history," in *IJCAI*, 2007.

[113] T. Jiang, H. J. Wang, and Y.-C. Hu, "Preserving location privacy in wireless lans," in *Mobisys*, 2007.

[114] J. Meyerowitz and R. R. Choudhury, "Realtime location privacy via mobility prediction: Creating confusion at crossroads," in *HotMobile*, 2009.