

# Towards Event Source Unobservability with Minimum Network Traffic in Sensor Networks\*

Yi Yang, Min Shao, Sencun Zhu, Bhuvan Uргаonkar, and Guohong Cao  
Department of Computer Science and Engineering, The Pennsylvania State University  
University Park, PA 16802, USA  
{yy5,mshao,szhu,bhuvan,gcao}@cse.psu.edu

## ABSTRACT

Sensors deployed to monitor the surrounding environment report such information as event type, location, and time when a real event of interest is detected. An adversary may identify the real event source through eavesdropping and traffic analysis. Previous work has studied the source location privacy problem under a *local* adversary model. In this work, we aim to provide a stronger notion: *event source unobservability*, which promises that a *global* adversary cannot know whether a real event has ever occurred even if he is capable of collecting and analyzing all the messages in the network at all the time. Clearly, event source unobservability is a desirable and critical security property for event monitoring applications, but unfortunately it is also very difficult and expensive to achieve for resource-constrained sensor networks.

Our main idea is to introduce carefully chosen dummy traffic to hide the real event sources in combination with mechanisms to drop dummy messages to prevent explosion of network traffic. To achieve the latter, we select some sensors as proxies that proactively filter dummy messages on their way to the base station. Since the problem of optimal proxy placement is NP-hard, we employ local search heuristics. We propose two schemes (i) Proxy-based Filtering Scheme (PFS) and (ii) Tree-based Filtering Scheme (TFS) to accurately locate proxies. Simulation results show that our schemes not only quickly find nearly optimal proxy placement, but also significantly reduce message overhead and improve message delivery ratio. A prototype of our scheme was implemented for TinyOS-based Mica2 motes.

## Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—Security and protection; C.2.1 [Computer-Communication

\*This work was supported in part by Army Research Office (W911NF-05-1-0270 and W911NF-07-1-0318), the National Science Foundation (CNS-0524156, CNS-0627382, CNS-0720456, and CAREER-0643906).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WiSec'08, March 31–April 2, 2008, Alexandria, Virginia, USA.  
Copyright 2008 ACM 978-1-59593-814-5/08/03 ...\$5.00.

**Networks**]: Network Architecture and Design—*Wireless communication*; K.6.5 [Computing Milieux]: Management of Computing and Information Systems—*Security and Protection*; K.4.1 [Computers and Society]: Public Policy Issues—*Privacy*

## General Terms

Security, Algorithms, Performance

## Keywords

Event Unobservability, Global Observer, Proxy, Minimum Traffic, Sensor Network Privacy

## 1. INTRODUCTION

Sensor networks bear a promising future in many important applications such as habitat monitoring, military surveillance, and target tracking. However, sensor networks are also confronted with many security threats such as node compromise, routing disruption and false data injection, because they normally operate in unattended, harsh or hostile environment.

Among all these threats, privacy is of special interest to us since it cannot be fully addressed by traditional security mechanisms, such as encryption and authentication. Consider a simple example of event reporting in a sensor network. When a sensor detects an event, it sends a message including event-related information to the base station. After this, the location of the event source has actually been leaked to the attacker (who may be passively monitoring the network), no matter how strong the data encryption key is. Furthermore, an attacker may find out more sensitive information: whether, when and where a particular event occurred, e.g., the appearing of an endangered animal in an asset monitoring sensor network [14, 22]. This can help the attacker in capturing the animal, an unfortunate occurrence.

Preserving event source location privacy, however, is a challenging task in sensor networks, which are characterized by limited resources in energy, computation, and communication. Hence, only lightweight, energy-efficient privacy-conserving mechanisms are affordable in sensor networks. Sensors typically have low-cost radio devices that employ standardized wireless communication technologies. The open architecture of the underlying sensor communication mechanisms enables an attacker to easily monitor or eavesdrop communications between sensors. Consequently, it is possible for a single attacker to monitor all the network traffic either by deploying his own simple sensors that cover the

whole deployment area [33] or by employing a powerful site surveillance device with hearing range no less than the network radius.

Despite its importance, source location privacy has not received due attention yet. A large number of anonymity techniques [5] designed for general networks are not appropriate to be used for sensor networks. This is not only because the privacy problem is different but also because these techniques are too costly to be employed. A few privacy-enhancing solutions [14, 24, 13, 30] have been proposed for sensor networks, but they assume relatively weak attack models. For example, in phantom routing [14, 24], an attacker has limited coverage, comparable to that of regular sensors. At any given time, only a single source is under the attacker’s consideration and the attacker tries to trace back to the source in a hop-by-hop fashion. When the attacker becomes more powerful, e.g., has a hearing range more than three times of the sensors, the scheme performs poorly since the capture likelihood may be raised to as high as 97%.

In this work, we aim to provide *event source unobservability* under a *global attack model*, where an attacker can hear and collect all the messages transmitted in the network at all the time. Event source unobservability promises that an attacker may neither discern the occurrence of a real event, nor find out the location of the real source. This is a stronger notion of privacy than traditional source location privacy that only hides the location of a real source. Under such an attack model, if all the packets in the network are real event packets, we are unlikely to achieve event source unobservability, because the transmission of a message, even encrypted, already indicates the occurrence of an event. Therefore, we devise schemes that introduce dummy traffic [10].

A *baseline* scheme based on such dummy traffic works as follows. Every node in the network sends out messages, either real or bogus, with intervals following a certain kind of distribution (e.g., constant rate or exponential). When a node detects a real event, it delays the transmission of the real event message such that the next inter-message interval follows the same distribution. Although this baseline scheme provides event source unobservability, it is also prohibitively expensive for sensor networks. The huge number of bogus messages not only consume the constrained energy of sensor nodes for transmissions, but also lead to high channel collision and consequently low delivery ratio of real event messages. Therefore, *it is our paramount goal to reduce the traffic while preserving event source unobservability*. To achieve this goal, we propose a Proxy-based Filtering Scheme (PFS) and a Tree-based Filtering Scheme (TFS). In PFS, some sensors are selected as proxies to collect and filter dummy messages from surrounding sensors. PFS greatly reduces the communication cost of the system by dropping many dummy messages before they reach the base station. In TFS, proxies are organized into a tree hierarchy. Proxies closer to the base station filter traffic from proxies farther away, thus the message overhead could be further reduced.

The message overhead imposed by these schemes is usually dependent on the locations of the proxies. Hence, based on local search heuristics we devise a proxy placement algorithm for each scheme to minimize the overall message overhead. Since real event messages may be delayed at the source due to the need to postpone their transmission, we also select suitable parameters for the buffers at the proxies

to reduce buffering delay while preserving event source unobservability. Our simulation results indicate that our schemes not only find nearly optimal proxy placement efficiently but also yield high delivery ratio and low bandwidth overhead, relative to the baseline scheme. A prototype of our schemes is implemented for TinyOS-based Mica2 motes [3], which consumes only about 400 bytes in the RAM space.

The rest of the paper is organized as follows. We first describe the problem and build up our model in Section 2. Then, Section 3 presents our PFS scheme and Section 4 presents the TFS scheme. Some issues are discussed in Section 5. After that, simulation and implementation results are presented in Section 6. Finally, we describe related work in Section 7 and conclude this paper in Section 8.

## 2. SYSTEM MODEL AND DESIGN GOALS

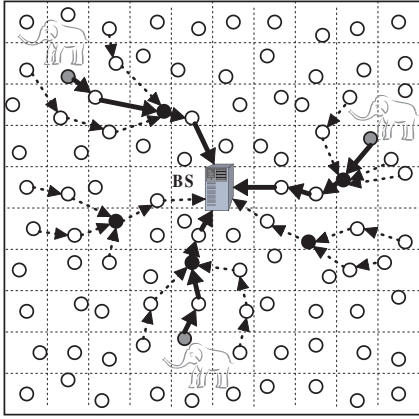
### 2.1 Network Model

As in [26], our system assumes that a sensor network is divided into cells (or grids) where each pair of nodes in neighboring cells can communicate directly with each other. A cell is the minimum unit for detecting events; a cell head coordinates all the actions inside a cell. Each cell has a unique integer id (in the range  $[1, n]$ , where  $n$  is the total number of cells) and every sensor node knows the cell in which it is located through its GPS or an attack-resilient localization scheme [21]. Also, we assume that a base station (BS) is located at the center of the network and works as the network controller to collect event data. An event report contains such information as the id of the detecting cell, the event type, and the detection time.

### 2.2 Attack Model

We assume that the adversary is *external*, *passive* and *global*. By external, we mean that the adversary will not compromise or control any sensors; by passive, we assume that the attacker does not conduct active attacks such as traffic injection, channel jamming and denial of service attack; by global, we assume that the adversary can collect and analyze *all* the communications in the network. Note that such a global attacker does not necessarily mean the attacker’s capability of detecting the occurrence of real events in any places of the network by himself, because (1) real event detection devices are often costly, whereas message collection devices are inexpensive and off-the-shelf; (2) real event detection devices such as animal-monitoring camera normally do not have sizes as small as regular sensors, so they are easy to be detected and destroyed. Although we do not consider sensor node compromises in the attack model, we will discuss this problem later in Section 5.3.

To be more specific, the adversary may launch the following attacks in our model. First, he may simply examine the content of an event message to see if it contains the source location id. Second, even if the message is encrypted, it is easy for him to trace back to the source of the message if the encrypted message remains the same during its forwarding, because the adversary is capable of identifying the immediate source of a message transmission. Third, he may perform more advanced traffic analysis including rate monitoring and time correlation. In a rate monitoring attack, the adversary pays more attention to the nodes with different (especially higher) transmission rates. In a time correlation attack, he may observe the correlation in transmission time between a



**Figure 1: Illustration of PFS. Blank circles and filled circles represent sources and proxies, respectively; dashed lines and solid lines denote bogus messages and real messages, respectively.**

node and its neighbor, attempting to deduce a forwarding path.

### 2.3 Design Goals

Providing event source unobservability under the global attack model is challenging. To prevent content-based analysis, we may encrypt all the packets during their forwarding, and also make all the packets in the network of the same length. However, these techniques cannot defend against rate monitoring and time correlation attacks.

To solve these traffic analysis attacks, we notice that there exist trade-offs between various performance and security metrics, such as privacy, delay, and communication cost. If all packets in the network are real event packets and every node reports and forwards a real event message immediately, it will be easy for a global attacker to trace back to the real source. Therefore, not only network-wide dummy traffic [10] but also delays in event reporting and forwarding have to be introduced at the nodes. Clearly, dummy traffic will significantly increase the network traffic, which is undesirable for sensor networks where communication overhead dominates the entire energy expenditure. To guarantee event source unobservability without causing the explosion of network traffic, in this paper our goal is to minimize the network traffic. Since it is hard to minimize the event report delay simultaneously, the proposed schemes are best suitable for applications in which a certain degree of delay could be tolerated.

## 3. PROXY-BASED FILTER (PFS) SCHEME

### 3.1 Scheme Overview

To employ dummy traffic to hide real events without incurring much message overhead, we select some sensors (in certain cells) as proxies to filter dummy messages before they reach the BS. The locations of these proxies are determined during network planning with the goal of minimizing aggregate network traffic. After network deployment, each proxy broadcasts a “hello” message with TTL (time to live) information that is large enough to reach every cell in the network. Every cell receiving these “hello” messages records

the proxy which is nearest to it as its default proxy. Every cell also sends back responses to its proxy so that each proxy knows which cells it serves for.

We assume each cell can establish a pairwise key with a proxy on the fly based on an appropriate keying scheme [20] and each proxy shares a key with the BS. When the network is in operation, each cell sends encrypted messages via unicast to its default proxy through a multi-hop routing protocol such as GPSR [15]. To satisfy our requirement of event source unobservability, the time intervals of these messages follow an exponential distribution (selecting other message generation patterns, such as a constant rate, does not affect our filtering schemes). When a cell detects an event, it postpones the transmission of the encrypted real event message to the next probabilistic interval, so that based on time analysis this message cannot be differentiated from dummy traffic.

Upon receiving a bogus message, a proxy performs filtering by discarding such a message. Upon receiving a real event message, the proxy reencrypts it (with a key shared with the BS) and forwards it towards the BS after an appropriate buffering time. In case of no real event message available, a proxy sends encrypted dummy ones instead. Note that proxies can differentiate dummy messages from real messages because they can properly decrypt the message using the corresponding pairwise key. If a proxy receives messages from other proxies, it just forwards them to the next hop. Figure 1 shows an example where PFS is employed for privately reporting elephant locations.

In summary, dummy messages are generated to hide real event messages. During this process, we minimize network traffic through optimal proxy deployment and preserve event source unobservability through appropriate filtering behavior inside proxies. Next, we describe optimal proxy placement in Section 3.2 and present proxy operations in Section 3.3. Finally, we analyze the security property of PFS in Section 3.4.

### 3.2 Proxy Placement

#### 3.2.1 Problem Statement

Deploying proxies in the right locations is crucial to the performance of our network. For example, if all the proxies are deployed close to each other, network traffic cannot be reduced effectively. Similarly, if all the proxies are placed far away from the BS, the number of bogus messages that can be filtered by proxies will be very limited. We consider the minimization of aggregate network traffic as the optimization criterion for our proxy placement. Aggregate traffic is defined as traffic rate  $\times$  message size  $\times$  number of hops (unit is byte  $\times$  hop/second). Since the sizes of all the event messages are the same, we only need consider traffic rate and total message transmission hops in the optimization problem.

In more detail, our proxy placement problem could be formalized in the following way. Suppose the set of all  $n$  cells in the network is denoted as  $V$  (i.e.,  $|V| = n$ ). Moreover,  $P$  is the set of proxy cells with size  $k$  ( $k \leq n$ ) and  $P$  is a subset of  $V$ . Since the closest proxy is the one that filters dummy messages for the cell, for a normal cell  $i$  in the network, its distance (i.e., number of hops) to the corresponding proxy could be expressed as:

$$d(i) = \min_{j \in P} d(i, j), \quad (1)$$

where  $d(i, j)$  is the distance between cell  $i$  and proxy  $j$  ( $1 \leq i \leq n, 1 \leq j \leq k$ ). Suppose that all the cells in the network send out event messages (dummy or real) following the same traffic rate  $r_{source}$  and the outgoing traffic rate from proxies is  $r_{proxy}$  (the relationship between  $r_{source}$  and  $r_{proxy}$  is determined by the buffering behavior inside the proxies). Our purpose is to minimize the following cost:

$$cost = r_{source} \cdot \sum_{i \in V} d(i) + r_{proxy} \cdot \sum_{j \in P} c(j). \quad (2)$$

That is,

$$cost = r_{source} \cdot \sum_{i \in V} \min_{j \in P} d(i, j) + r_{proxy} \cdot \sum_{j \in P} c(j), \quad (3)$$

where  $c(j)$  is the distance from proxy  $j$  to the BS.

A solution to the proxy placement problem consists of the number of proxies needed and their locations that minimize the cost described above. This problem is NP-hard, because one proxy's even one-step of position change may cause recalculations for all cells, which cannot be effectively solved in polynomial time. The well-known facility location problem [29] reduces to it. We adapt heuristics based on localized search [6, 19] to efficiently solve the proxy placement problem. Next, based on this idea, we devise proxy placement algorithm that can find approximately optimal locations for proxies, given any  $r_{source}$  and  $r_{proxy}$ .

---

#### Algorithm 1 Proxy Placement Algorithm in PFS

---

**Input:** a cell-based network topology with node set  $V$ ; the total number of nodes  $n$ ;

**Output:** a set of proxies  $P$ ;

**Procedure:**

```

1:  $P \leftarrow \Phi$ ;  $P' \leftarrow \Phi$ ;  $\{cost(\Phi) = \infty\}$ 
2: for  $k \leftarrow 1$  to  $n - 1$  do
3:    $placement(k)$ ;  $\{\text{Update } P'\}$ 
4:   if  $cost(P') < cost(P)$  then
5:      $P \leftarrow P'$ ;
6:   end if
7: end for
8: return  $P$ ;
9:
10:  $placement(k)$ 
11:  $P'[0] \leftarrow BS$ ;
12: for  $i \leftarrow 1$  to  $k$  do
13:    $P'[i] \leftarrow i$ ;  $\{\text{Initialize } P'[0] \dots P'[k]\}$ 
14: end for
15: for  $\forall i \in P'$  and  $\forall j \notin P'$  and  $i, j \in V$  do
16:    $P'' \leftarrow P' - i + j$ ;  $\{\text{Swap } i \text{ and } j\}$ 
17:   if  $cost(P'') < cost(P')$  then
18:      $P' \leftarrow P''$ ;
19:   end if
20: end for;  $\{\text{Loop ends after we try all the combinations of } i \text{ and } j\}$ 

```

---

### 3.2.2 Proxy Placement Algorithm

The details of our proxy placement algorithm are presented in Algorithm 1. Given  $k$ , the number of proxies to be deployed, we begin with a random initial set  $P'$  of size  $k$  (e.g., the first  $k$  nodes) with the BS as a default proxy. Our algorithm proceeds in steps, in each of which we try to swap a node in set  $P'$  with a node that is currently not in set  $P'$ , if such a swap would reduce the aggregate network traffic. If a swap succeeds, then we update the set  $P'$  accordingly. We repeat this process until no more swaps are possible. At

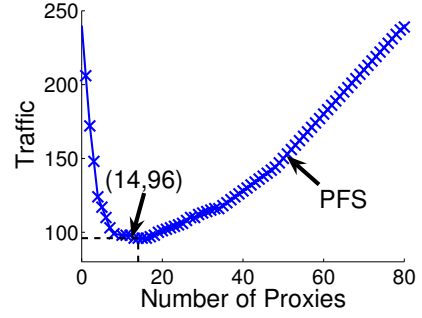


Figure 2: The optimal number of proxies in PFS.

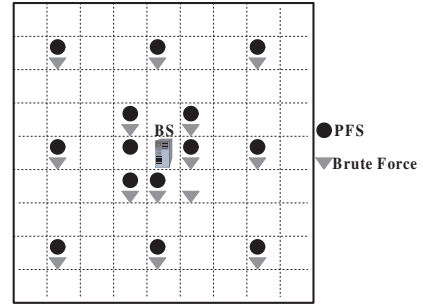


Figure 3: The optimal proxy placement.

this point, the cost reaches a local minimum. Note that for any  $k$  this process is guaranteed to converge since each swap results in a reduction in the cost of the solution which is lower bounded by a positive value corresponding to the optimal solution for  $k$ . We vary  $k$  from 1 to  $n$  and record the set  $P$  with the minimum cost. After we obtain  $P$ , the size of  $P$  is the number of proxies in our deployment and cells corresponding to the set  $P$  are those where we place these proxies.

We have the following results on the time complexity of Algorithm 1. For the average case, it is hard to analyze the number of iterations in each  $placement(k)$ . Similar to [25], we resort to experiment and find that the average-case complexity of this algorithm tends to be  $O(n^4)$ .

In the worst case, the initial set  $P'$  is shifted by a maximum distance. For example, suppose the initial set  $P' = \{BS, 1, 2, \dots, k\}$  and the algorithm returns  $\{BS, n - k + 1, \dots, n\}$  ( $BS$  is a default proxy). Then, each element in  $P'$  is shifted by a distance  $n - k + 1$ , one step each time. Take element  $k$  in the initial set for an instance, the shifting sequence will be  $k, k + 1, \dots, n$ . For the first shift from  $k$  to  $k + 1$ , we need to compare  $k + 1$  with all the elements in  $P'$  which includes  $1, 2, \dots, k$  and the last attempt succeeds. Totally  $k$  swaps have been tried. For the second shift from  $k + 1$  to  $k + 2$ , first, we compare  $k$  with all the elements in  $P'$  including  $1, 2, \dots, k - 1, k + 1$  and all attempts fail (the last attempt between  $k$  and  $k + 1$  can be ignored, but we count it here for simplicity). Then, we compare  $k + 2$  with all the elements in  $P'$  which includes  $1, 2, \dots, k - 1, k + 1$  and the last attempt succeeds. Hence, totally  $2k$  swaps are attempted. Similarly, this process continues until the last shift from  $n - 1$  to  $n$  which has  $(n - k + 1)k$  swaps. Therefore, for each element like  $k$  in set  $P'$ , there are

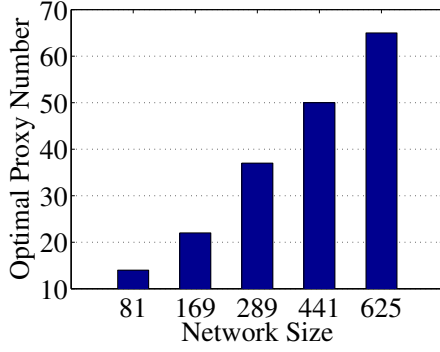


Figure 4: The impact of network scale on the optimal proxy number in PFS.

$k \cdot \sum_1^{n-k+1} i = O((n-k)^2)$  swaps being attempted. To shift the initial set of  $P'$  to its final set, the total number of swaps is  $\sum_{i=1}^k i(n-k+2)(n-k+1)/2 = O(k^2(n-k)^2)$ . Since the time to execute each swap is  $O(k(n-k))$ , the time for one *placement(k)* is  $O(k^3(n-k)^3)$ . We try all the possible  $k$  from 1 to  $n$ . Hence, the worst-case time complexity of Algorithm 1 is  $\sum_{k=1}^n O(k^3(n-k)^3) = O(n^7)$ , which is much less than that of the brute force method (where we try all the possibilities to finally find an optimal placement) in  $O(n!)$  (when  $n \geq 2$ ).

Through employing local search heuristics, Algorithm 1 can be completed in polynomial time. Note that Algorithm 1 is executed during network planning by network controller who has sufficient computation and energy resources, with the results being preloaded in each sensor's memory before node deployment. Individual sensors do not need to compute this separately.

### 3.2.3 Performance Evaluation

We conducted simulations to evaluate Algorithm 1. In the simulation, we setup a network with  $n = 81$  and continuously run *placement(k)* with  $k = 0, \dots, 80$ . As shown in Figure 2, the minimum cost obtained by our algorithm is 96 when  $k = 14$ . It is as accurate as the globally optimal value returned by the brute force method. We also evaluate the efficacy of our algorithm in determining the locations where the proxies should be deployed through simulation. As can be seen in Figure 3, positions of proxies returned by our algorithm and the optimal positions obtained by the brute force method are almost the same. Hence, we believe that our algorithm can quickly find a placement solution that closely approximates the optimal solution.

We also check the impact of network scale on the number of proxies chosen by our placement algorithm. The results are shown in Figure 4. When the network sizes are 81( $9 \times 9$ ), 169( $13 \times 13$ ), 289( $17 \times 17$ ), 441( $21 \times 21$ ), and 625( $25 \times 25$ ), the numbers of proxies determined by our algorithm are 14, 22, 37, 50, and 64, respectively. These results will also be used in our following performance evaluation in Section 6. Apparently, the optimal number of proxies increases with a larger network scale.

## 3.3 Proxy Operations

Upon receiving a message, a proxy performs the following operations to reduce the network traffic while preserving

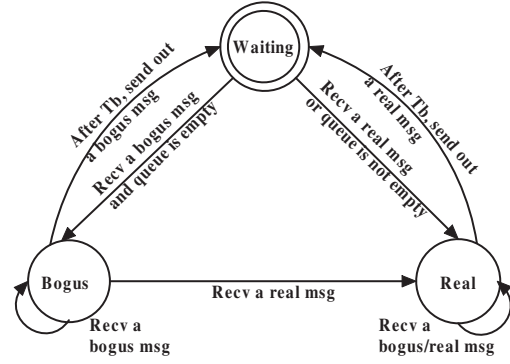


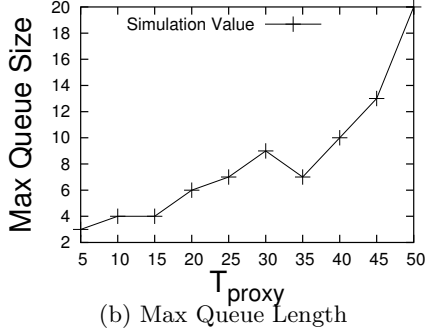
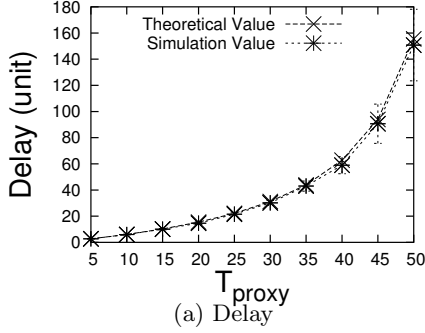
Figure 5: State transitions of proxies (there are three states: waiting, bogus, real).

event source unobservability:

- First, the proxy decrypts the message so that the proxy can differentiate real event messages from bogus ones;
- Second, the proxy drops the message immediately if it is a bogus message. If on the other hand the message corresponds to a real event, the proxy re-encrypts the decrypted message;
- Third, the proxy puts this re-encrypted real event messages into its message buffer. After a constant time, a message, either bogus or real, will be sent out from the proxy node.

Let us discuss these internal operations of a proxy in more detail. These operations can be understood using the state transition diagram of a proxy shown in Figure 5. As shown, a proxy is initialized to be in the *Waiting* state. Upon receiving a message, the proxy first decrypts the packet. If the received message is bogus, the proxy drops this message immediately. If its message buffer is now empty, the proxy changes its state to *Bogus*. Otherwise, if the received message is real then the proxy saves the real message in its message buffer. Whenever there are real messages in the message buffer to be sent out, the proxy switches its state to *Real*. Additionally, if the proxy is in state *Bogus*, it will change to state *Real* if a real message is received and remain in state *Bogus* if a bogus message is received. On the other hand, if the proxy is in state *Real*, it will remain in state *Real* regardless of the type of message received as long as there is at least one real message in the buffer.

Recall that the goal of each proxy is to ensure that the outgoing traffic conforms to a rate of  $r_{proxy}$  requests per time unit. Recall also that in this work we consider outgoing traffic with requests (messages) equally spaced in time. That is, a proxy emits a message once every  $T_{proxy} = \frac{1}{r_{proxy}}$  time units. To achieve this, the proxy repeats the following process over non-overlapping and successive *intervals* of duration  $T_{proxy}$  each. During each of such an interval, the proxy adds all the received real messages to its buffer. If the buffer is full, then the new incoming real messages are dropped (in our real implementation for Mica2 sensor nodes, such messages are buffered in the large flash memory to avoid dropping). At the end of each interval, the proxy sends out a message depending on what state it was in: if it is in state *Bogus*, a bogus message is sent out; if it is in

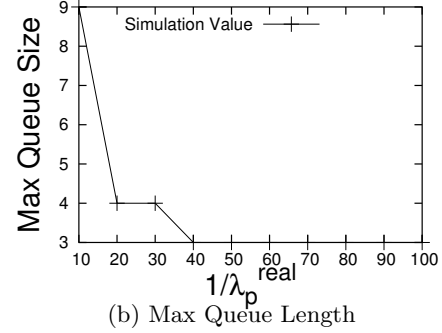
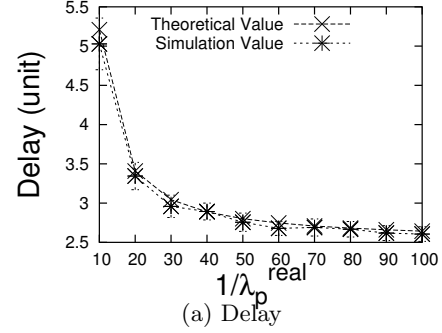


**Figure 6: Delay and max queue length under  $\lambda_{\mathcal{P}}^{real} = 1/60$  per time unit.**

state *Real*, a real message is sent out (FIFO ordering is used in case multiple real messages have buffered up).

While the introduction of buffering at the proxies enables the network to achieve its goals of event source unobservability (as will be shown formally in Section 3.4), it degrades performance by introducing additional delays in the delivery of real event messages. *How can we estimate the delay in message delivery resulting from PFS?* We use queuing theoretic model of a proxy to conduct a simple analysis of this delay.

Let  $\lambda_{\mathcal{P}}$  denote the rate at which messages arrive at the proxy  $\mathcal{P}$ . Recall that our design makes source nodes generate traffic with exponentially distributed inter-arrival times with a rate  $r_{source}$  that is identical across all the source nodes. Therefore,  $\lambda_{\mathcal{P}} = n_{\mathcal{P}} \cdot r_{source}$ , where  $n_{\mathcal{P}}$  is the number of source nodes associated with the proxy  $\mathcal{P}$ . Since the processing time at the proxy (in the millisecond level for a Mica node, which is mainly due to decryption/reencryption) is significantly smaller than  $T_{proxy}$ , we make the simplifying assumption that this process time can be ignored. In other words, the delay caused in the delivery of a real message is essentially the sum of the delays caused by the buffering at the source node where the event occurred and at the proxy that the message passes through on its way to the BS. Finally, we assume that the fraction  $f_{\mathcal{P}}^{real}$  of the messages arriving at  $\mathcal{P}$  is real. We can now view the proxy  $\mathcal{P}$  as an M/G/1 [17] queuing system since: (i) the arrival process is Poisson with a rate  $f_{\mathcal{P}}^{real} \cdot \lambda_{\mathcal{P}}$  and (ii) the time a real request spends in the message buffer may be considered as a random variable  $s_{\mathcal{P}}$  representing the “service time” of the request. Note that the notion of servicing in this queue is an abstract one and the time spent waiting in the message



**Figure 7: Delay and max queue length under  $T_{proxy} = 5$  time units.**

buffer is being modeled as the service time in a hypothetical server. Also, our assumption above implies that the dummy traffic arriving at  $\mathcal{P}$  plays no role in our analysis. Therefore, a *request* will be meant to signify a *real request* in the rest of the analysis.

We can derive some useful properties of the random variable  $s_{\mathcal{P}}$  using standard queuing theory. Let  $p_{\mathcal{P}}$  denote the probability that a newly arriving request finds the queue representing  $\mathcal{P}$  idle. The well-known PASTA property (“Poisson Arrivals See Time Averages”)[17] states that for queuing systems with Poisson arrivals, the fraction of requests finding the queue idle upon arrival is exactly the same as the fraction of time the system is idle. This implies that  $p_{\mathcal{P}}$  is equal to the probability that the queue is idle. Clearly, requests that arrive to the queue when it is idle will experience an average service time  $\frac{T_{proxy}}{2}$  because in this case service times follow a uniform distribution  $U[0, T_{proxy}]$  (and hence the variance is  $T_{proxy}^2/12$ ); other requests will experience a service time  $T_{proxy}$  since requests emerge from  $\mathcal{P}$  at the rate  $r_{proxy}$ . Therefore, we have,

$$E[s_{\mathcal{P}}] = p_{\mathcal{P}} \frac{T_{proxy}}{2} + (1 - p_{\mathcal{P}})T_{proxy}; \quad (4)$$

$$E[s_{\mathcal{P}}^2] = p_{\mathcal{P}} \frac{T_{proxy}^2}{3} + (1 - p_{\mathcal{P}})T_{proxy}^2 \quad (5)$$

In Formula (5)  $\frac{T_{proxy}^2}{3}$  is derived from the variance and the mean of the service time when the queue is idle. Furthermore, queuing theory [17] gives us,

$$p_{\mathcal{P}} = 1 - f_{\mathcal{P}}^{real} \cdot \lambda_{\mathcal{P}} \cdot s_{\mathcal{P}} \quad (6)$$

Combining these equations, and denoting  $f_{\mathcal{P}}^{real} \lambda_{\mathcal{P}}$  as  $\lambda_{\mathcal{P}}^{real}$ ,

we have,

$$E[s_{\mathcal{P}}] = \frac{T_{proxy}}{2 - \lambda_{\mathcal{P}}^{real} T_{proxy}}; \quad (7)$$

$$E[s_{\mathcal{P}}^2] = (1 - \lambda_{\mathcal{P}}^{real} E[s_{\mathcal{P}}]) \frac{T_{proxy}^2}{3} + \lambda_{\mathcal{P}}^{real} E[s_{\mathcal{P}}] T_{proxy}^2 \quad (8)$$

The average delay  $d_{\mathcal{P}}$  experienced by a message at the proxy  $\mathcal{P}$  is then given by the following result known for the average sojourn time of a request in our M/G/1 queuing system,

$$d_{\mathcal{P}} = E[s_{\mathcal{P}}] + \frac{\lambda_{\mathcal{P}}^{real} (E^2[s_{\mathcal{P}}] + E[s_{\mathcal{P}}^2])}{2 \cdot (1 - \lambda_{\mathcal{P}}^{real} E[s_{\mathcal{P}}])} \quad (9)$$

To verify our theoretical results we use CSIM [1] to simulate this queueing model. In our simulation setup,  $T_{proxy}$  changes from 5 to 50 time units and  $\lambda_{\mathcal{P}}^{real}$  changes from 1/10 to 1/100 per time unit. We obtain the queuing delay as shown in Figure 6(a) and Figure 7(a). The error-bars show the 95% confidence interval of the simulation result. From these figures, we can see that our theoretical values and simulation results match well. In Figure 6(a), if the incoming rate  $\lambda_{\mathcal{P}}^{real}$  is fixed, then the average delay increases as the interval  $T_{proxy}$  increases, because of the longer queue length and buffer time. On the other hand, in Figure 7(a), if the interval  $T_{proxy}$  is fixed, then the average delay decreases as the incoming rate  $\lambda_{\mathcal{P}}^{real}$  decreases, since the buffer is less occupied.

We also use simulation to check the queue length. Our simulation results in Figure 6(b) and Figure 7(b) show the maximum queue lengths under different simulation settings. These results could be used to provide guidance for allocating proper buffer size. If we select a larger interval  $T_{proxy}$ , then the buffer size should also be increased accordingly. Also, if the incoming rate  $\lambda_{\mathcal{P}}^{real}$  is lower, then a smaller buffer size will be enough for our application.

### 3.4 Security Analysis

According to [11], we have the following definitions on event source unobservability.

**DEFINITION 1.** If for each possible observation  $O$  that an attacker  $A$  can make, the probability of an event  $E$  is equal to the probability of  $E$  given  $O$ , that is:  $\forall O, P(E) = P(E|O)$ , then  $E$  is called **unobservable**.

**DEFINITION 2.** A system has the property of **event source unobservability** if any event  $E$  happening in this system is unobservable:  $\forall E, \forall O, P(E) = P(E|O)$ .

Next, we prove that our system can achieve the property of event source unobservability.

**THEOREM 1.** PFS has the property of event source unobservability.

**PROOF.** (Sketch)  $P(E) = P(E|O)$  in Definition 1 means that event  $E$  and observation  $O$  are independent ( $P(E \cap O) = P(O) \cdot P(E|O) = P(O) \cdot P(E)$ ). Therefore, according to the above definitions, if we want to prove that PFS has the property of event source unobservability, then we need prove that any event  $E$  is independent of the observation  $O$  that the attacker  $A$  makes.

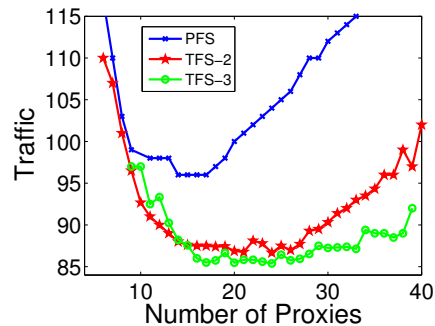


Figure 8: Improvement of TFS over PFS.

Let us first consider all the possible observations that an attacker can make from the system. The adversary can observe messages being generated at each cell with intervals following a certain probabilistic distribution. However, as the messages are encrypted and of the same length, the attacker cannot distinguish the real ones from the dummy ones. These messages are then relayed on visible multi-hop paths to proxies. Proxies drop and delay messages, but the attacker does not know which messages are dropped and which are forwarded by proxies due to message reencryption and constant-rate eviction. The outgoing messages from proxies are finally forwarded to the BS in always the same way. Thus, even with all these observations, the attacker cannot gain any additional information on real events. The occurrence of a real event  $E$  is independent of attacker's observation  $O$ . Therefore, every real events are unobservable for the attacker. According to Definition 2, PFS has the property of event source unobservability.  $\square$

## 4. TREE-BASED FILTER SCHEME (TFS)

If the number of proxy nodes is large enough, further reduction in the dummy traffic is possible by allowing messages to be filtered at multiple proxies on their way from source nodes to the BS. Note that in PFS, even though a message may traverse through multiple proxies, it is filtered only at the default proxy of the cell that this message originates from. Building upon this core idea, we propose a *tree-based filtering scheme* (TFS) in which the proxies in our network are organized in the form of a tree rooted at the BS. Proxies in TFS, thus, form a *hierarchy* with each proxy having a *parent* node and possibly multiple *child* nodes. With the resulting multi-level filtering, we expect lower network traffic because more dummy messages will be dropped before they reach the BS. The reduction in traffic due to TFS, however, will come at the expense of increased latency of real event delivery since each message may now incur buffering-induced delays at multiple proxies on its way to the BS. This trade-off between traffic and latency is central to the research issues involved in the design and efficacy of TFS.

### 4.1 Hierarchical Proxy Placement

Randomly picking up proxies might end up assigning more proxies to some paths than others, which may limit the overall filtering efficacy of TFS. Therefore, similar to our approach in PFS, we devise a heuristic based on localized

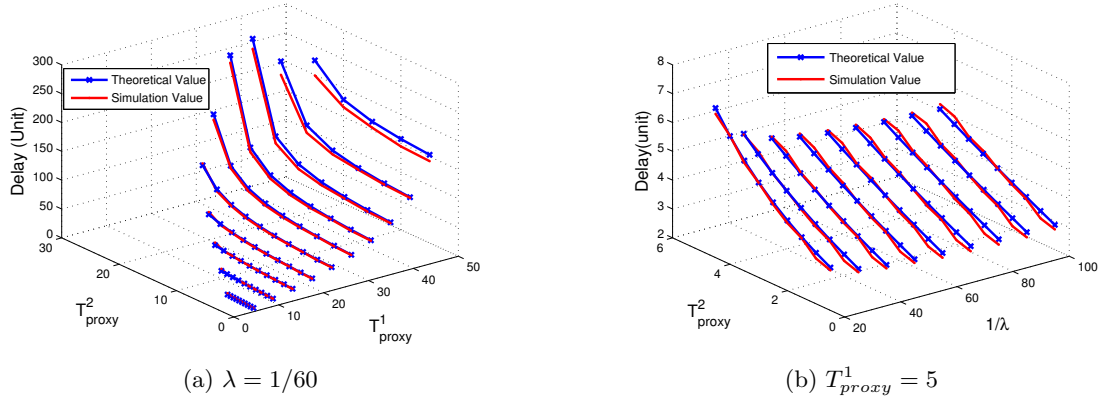


Figure 9: Delay in TFS under different  $T_{proxy}$  and  $\lambda$  (tree level  $l = 2$ ).

search to derive effective proxy placement in an efficient way. Through experiments, we found that if there are more constraints on the proxy placement then the result of the algorithm is better (i.e., closer to the globally optimal solution). Hence, in the algorithm we specify the total number of levels for the proxy tree. Another benefit of this mechanism is that by assigning a small number of levels we can also reduce the real event report latency since real event messages need go through fewer levels of proxies before they reach the BS.

We adapt Algorithm 1 to implement the proxy placement algorithm for TFS- $l$  (the proxy level  $l$  is specified as a parameter). We do not present its details since these two algorithms follow the same local search heuristics so the changes are not much. Different from Algorithm 1 in PFS, here the set  $P'$  has an internal hierarchical structure. Because of this, an *inner-swap*, which works by exchanging the positions of proxies within the current set, might reduce the cost too. The proxy number  $k$  is iterated from 1 to  $n$ . For each given  $k$ , we try all the combinations of proxy numbers in every level and record the set with the minimum cost as  $P$ . The algorithm keeps on running until no more swap or inner-swap that could reduce the cost exists, so that the total cost reaches its local minimum.

We also briefly analyze the time complexity of this new algorithm. Analyzing the time complexity for the average-case is difficult since quantifying the total number of iterations in each *placement*( $k$ ) is unwieldy. Through repetitive experiments and curve fitting, we find that the average time complexity of this algorithm is about  $O(n^{3+l})$ . In the worst case, the time complexity of this algorithm is  $O(n^{6+l})$ . When  $l = 1$ , this time complexity is exactly the same as that of Algorithm 1, which verifies the correctness of our analysis.

We use simulations to compare the traffic generated by the two schemes. With  $n = 81$ , for proxy number  $k$  ranging from 5 to 40, we check the trends of traffic under different number of tree levels (i.e.,  $l = 1, 2, 3$ , respectively). The improvement of TFS over PFS in overall traffic is evident in Figure 8. However, we notice that event latency may be increased in TFS since each message may go through several proxies, each of which has a buffering period.

## 4.2 Multi-level Buffering Delays

In the hierarchical organization of proxies employed in TFS, the proxies at each level have an interval that deter-

mines the rate at which traffic emerges from them. For  $l$  levels in the hierarchy, we use the notation  $T_{proxy}^1, \dots, T_{proxy}^l$  to represent the intervals, with level-1 corresponding to the leaf nodes and level- $l$  corresponding to the children of the BS. The analysis in PFS extends directly to the “leaf-level” proxies in the TFS hierarchy of proxies. For intermediate proxies, however, the situation becomes more complicated. The incoming traffic to an intermediate proxy  $\mathcal{P}_{int}$  consists of: (i) the outgoing traffic from one or more other proxies that are its “children” in the tree imposed on the proxies by the placement algorithm, and (ii) messages from the cells that  $\mathcal{P}_{int}$  is in-charge of. The outgoing traffic from a proxy described in PFS was designed to follow a constant rate  $r_{proxy}$ , whereas the messages received from a cell arrive according to a Poisson process with a rate of  $r_{source}$ . As a result, the incoming traffic for the intermediate proxies in TFS does not conform to a Poisson process (as it did for the proxies in PFS and continues to do for the “leaf” nodes in TFS). *This does not compromise the privacy guarantees we wish to provide.* However, it does mean that the applicability of our M/G/1 queueing model for determining the buffering-induced delay at such intermediate proxies becomes questionable. This can be addressed easily by re-designing the buffering mechanism in our proxies such that they emit traffic that conforms to a Poisson process instead of generating the deterministic traffic described in PFS. That said, we note that the estimation of delay is not a key concern in our current research since our focus is on providing privacy for relatively delay-tolerant applications. Therefore, we continue with a proxy design that generates output traffic deterministically with fixed time-gaps between messages and use the M/G/1 model as an approximate representation of a proxy. We evaluate this approximation empirically below.

We conduct simulations to check the relationship of real event report delay with the buffer intervals for each level as well as the traffic arrival rate. The real event report delay under consideration is an aggregate delay since each message may go through several proxies, each of which has a buffering period. In Figure 9(a), we fix the traffic rate arriving at an illustrative 2-level hierarchy consisting of proxies  $\mathcal{P}_1$  and  $\mathcal{P}_2$  ( $\mathcal{P}_1$  is the only child of  $\mathcal{P}_2$  and the only source of input traffic to  $\mathcal{P}_2$  in this example) to be  $1/60$  per time unit and observe the change of delay introduced by the buffering at  $\mathcal{P}_1$  and  $\mathcal{P}_2$  with different  $T_{proxy}^1$  and  $T_{proxy}^2$ . Clearly, delay increases as

$T_{proxy}^1$  or  $T_{proxy}^2$  increases. In Figure 9(b), we fix  $T_{proxy}^1 = 5$  time units and present the change of delay with the traffic arrival rate, with the results showing that delay decreases as the mean arrival rate decreases.

### 4.3 Security Analysis

**THEOREM 2.** TFS has the property of event source unobservability.

The correctness of this theorem can be proved based on the similar arguments we used to prove Theorem 1. We omit the details due to space limit.

## 5. PRACTICAL CONSIDERATIONS

In this section, we address two key sets of issues that must be addressed for effective realization and deployment of the privacy-preserving schemes developed in this paper.

### 5.1 System Parameters

Choosing appropriate values for the source traffic generation rate and the buffering intervals is very important to balance privacy, delay, and message overhead according to the needs of our applications. We notice that under the purpose of achieving event source unobservability, delay and overhead are actually tightly related to each other. How to choose parameters depends on the relative criticality of these two requirements in our application.

The first parameter to be decided is the source traffic rate (consisting of real and dummy messages)  $r_{source}$ . If the dummy traffic rate is too high, it will unnecessarily cause high message overhead; if it is too low, real event messages will experience high transmission latency at the sources. It is desirable to have this rate as close as possible to the average real event message rate. We believe that in practice it is not difficult to determine an appropriate  $r_{source}$  based on historical information about event generation at the sources.

The more interesting issue concerns the buffering interval  $T_{proxy}$  employed by the proxies in PFS (or buffering intervals, one per level, in TFS; we conduct our discussion in the context of PFS but the ideas extend to TFS as well). Since  $T_{proxy}$  determines the rate at which messages leave a proxy, we need to ensure that it is chosen such that the this departure rate exceeds the aggregate rate at which *real* messages arrive at the proxy. Otherwise, real event messages may be dropped at proxies. That is, we require  $\forall \mathcal{P}, agg(\lambda_{\mathcal{P}}^{real}) < \frac{1}{T_{proxy}}$ . This gives us,

$$\frac{1}{T_{proxy}} > agg_{\mathcal{P}}(\lambda_{\mathcal{P}}^{real})$$

The choice of  $T_{proxy}$  concerns balancing the trade-off among the following opposing trends. Picking a small  $T_{proxy}$  reduces the probability of messages being dropped due to the finite buffer at a proxy overflowing. Also, smaller values of  $T_{proxy}$  result in smaller values of additional delay introduced by the buffering mechanism at a proxy. On the other hand, picking a large  $T_{proxy}$  reduces the overall traffic by generating fewer dummy messages at the proxies. The choice of  $T_{proxy}$  will depend on the relative criticality of these opposing requirements. As already discussed, our queuing model serves as a reasonable predictor of the delay caused by the choice of a particular  $T_{proxy}$ . Also, given  $T_{proxy}$  and other

relevant parameters, we may easily calculate the communication cost  $cost$  defined in Formula 3. Thus, we may adjust  $T_{proxy}$  in generating acceptable delay and communication cost.

### 5.2 Role Shifting among Proxy Nodes

The proxy nodes do more work than other sensor nodes: (i) reception and filtering of dummy messages, (ii) decryption /re-encryption of real event messages, and (iii) buffering and forwarding of real event messages. Therefore, proxy nodes drain their energy resources faster than normal nodes. To prolong the lifetime of the network, after a certain time period, new proxies may be selected to replace old ones. Cell head may choose another sensor node in the same cell to act as proxy, i.e., proxy's cell-based position does not change. Another choice is to rotate the proxies' positions in the network, determined by BS. The new proxy notification message that includes the map of new proxies could be broadcasted to the entire network. In this case we need to deal with the issue of *hand-off*. After a shift, real event messages may stay in the buffers of some old proxies. A reasonable solution is to have the old proxies behave like regular nodes and forward all the real event messages to their closest new proxies.

### 5.3 Insider Attacks

Although we did not consider sensor node compromises in our attack model of Section 2.2, we notice that our schemes are robust to this kind of insider attacks, due to the following reasons. Compromised source sensors do not know which packets will be dropped by proxies and compromised forwarding sensors do not have appropriate decryption keys. Also, the injected false packets from these sensors to launch denial of service attack will be dropped by proxies. Therefore, these compromised sensors could not do much to break our scheme. Compromised proxy sensors may be a problem because they know which packets are from real sources, but this kind of sensor network privacy related node compromises have been discussed in our previous work [27].

## 6. PERFORMANCE EVALUATION

In this section, we use simulations to compare the performance of PFS and TFS schemes with the baseline scheme (i.e., a scheme without proxies, in which every sensors send either real or bogus messages following a certain interval, as described in Introduction). After that, implementation results are presented.

### 6.1 Simulation Setup

The simulation is based on GloMoSim [2]. In the simulation, 625 sensor nodes are deployed in a 1000m×1000m area. For each sensor node, the transmission range is 50m. The BS is located at the center of the field. Five cells (sources) are randomly selected to generate real event messages and other cells generate bogus messages, with intervals following exponential distributions. For real event generation, we consider two cases: a heavy-traffic case and a light-traffic case, with 10s and 400s as the mean of inter-message intervals respectively, to simulate various situations that possibly happen. For example, if the animal stays still for rest, the real event generation rate from detection sensor may be low; Otherwise, if the animal moves fast, then the real event message generation rate will be adjusted to be higher. In addition,

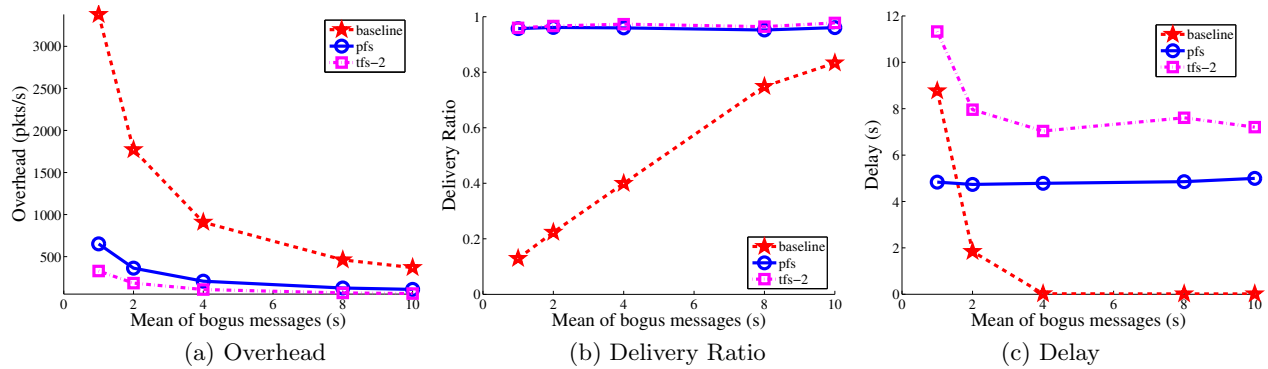


Figure 10: Performance under different bogus message generation rate (heavy-rate real events).

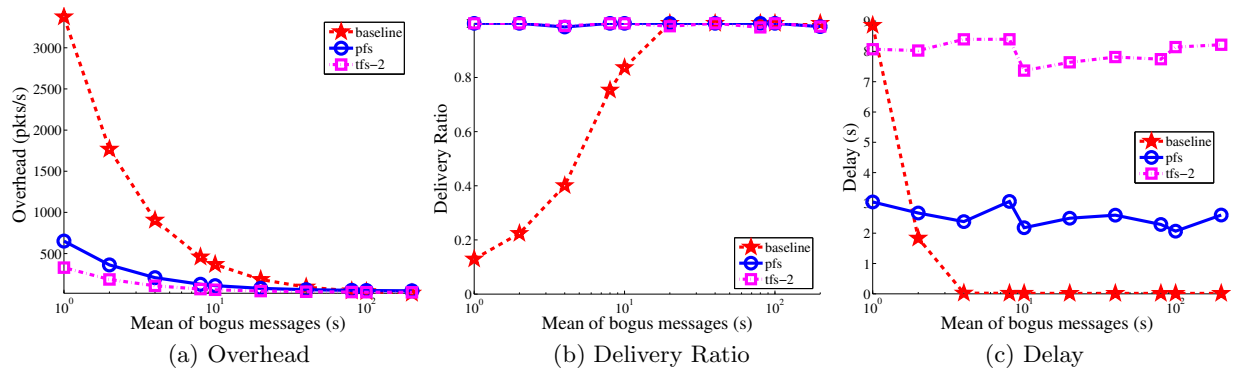


Figure 11: Performance under different bogus message generation rate (light-rate real events).

the mean of dummy traffic varies from 1s to 200s. The buffer interval  $T_{proxy}$  is set to 5s, and the buffer size is 10 packets. For TFS, we set the tree level as two.

During our evaluation, we use three metrics: *message overhead*, *packet delivery ratio*, and *delay*. Message overhead is defined as the product of total number of messages and number of hops they traversed. Packet delivery ratio is the percentage of real event messages that successfully reach the BS. Delay is the time difference between the time when a real event message is generated and when it reaches the BS.

## 6.2 Simulation Results

Figure 10 and Figure 11 show the impact of different bogus message generation rate (in terms of mean of bogus message intervals) to the message overhead, the packet delivery ratio and the delay. They correspond to the cases of heavy-rate real events and light-rate real events, respectively.

From Figure 10(a) and Figure 11(a), we can see that in all three schemes the message overhead increases when the bogus message generation rate increases (as the means decrease in the x-axis of these figures). However, the message overhead of PFS and TFS increases much slower compared to the baseline scheme. Among them, TFS incurs the overhead below 1/10 that of the baseline scheme, and is hence the most bandwidth-efficient.

Figure 10(b) and Figure 11(b) show that packet delivery ratio decreases when the bogus message generating rate increases. This is because the chance of MAC layer collision increases with dummy traffic rate. Since we did not run any end-to-end reliable protocol, some messages including the

real event messages, may be lost. (Indeed, end-to-end reliability is likely to make the situation worse.) The figures also show that the packet delivery ratio of the baseline scheme is very low (less than 20%) when the bogus message generating rate is high, but both PFS and TFS have the delivery ratio close to 100%.

Figure 10(c) and Figure 11(c) indicate, without much surprise, that the delay of PFS and TFS is normally much higher than that of the baseline scheme. The delay of TFS is about twice of PFS because here the TFS tree is two-level. We note in some cases, as shown in Figure 11(c), when the network traffic becomes very heavy (mean of bogus messages is 1s), the delay of PFS is actually lower than that of the baseline scheme due to high collision in the baseline scheme.

In summary, both PFS and TFS are good choices because of high packet delivery ratio and low message overhead, whereas the baseline scheme normally incurs low delay.

## 6.3 Prototype Implementation

To study the practicality of our schemes, we implement a prototype of our PFS scheme on top of TinyOS [4] for Mica2 motes. Since a mote has only 4KB RAM space, it is not always possible to buffer all the real messages. To avoid message dropping in the case of burst events, in our implementation overflowed packets are cached in the 512-KB flash memory, which is the event logging space available to Mica2 motes. Whenever the buffer has spare space, a cached message is moved to the end of the buffer immediately. An outside observer would not see the caching and moving operation inside a mote.

Our code consumes 13.7KB (out of 128KB) in the program

memory and 399B (out of 4KB) in the data memory. We test the queuing behavior of a mote under various message arrival rates, and find the results agree with what we may get through queuing analysis and simulations.

## 7. RELATED WORK

Since Chaum's seminal work in 1981 [8], so far hundreds of papers [5] have been concentrated on building, analyzing, and attacking anonymous communication systems. Due to space limit, we can only discuss those most relevant ones in both wired networks and wireless networks.

### 7.1 Anonymous Communications in Wired Networks

The work that is most relevant to ours is mix and mix cascade [8]. Mix is a node that hides the correspondences between its input messages and its output messages in a cryptographically strong way. To achieve this goal, a mix changes the appearance by encryption and padding. It also changes the flow of messages by collecting multiple messages, reordering them, and then flushing them in a batch, ensuring the timing of messages does not leak any linking information. Many mix-based systems have been designed later, such as web mixes [7], Mixmaster [23] and Stop-and-Go mix [16]. Onion routing [12] is the equivalent of mix networks but in the context of circuit-based routing.

In our schemes, a proxy node is similar to a traditional mix in that it changes the appearance (by decryption/reencryption) and flow (by delaying) of real event messages. It differs from a traditional mix in that (1) it does not perform message reordering, that is, real event messages are forwarded in a first-come-first-out basis; (2) it has only one fixed output link; and (3) it drops some input messages.

### 7.2 Anonymous Communications in Wireless Networks

Recently several on-demand protocols have been proposed for anonymous routing in mobile ad hoc networks, including ANODR [18], ASR [32], and MASK [31]. They use techniques such as pseudonym and broadcast, and most of them are based on public key cryptography. Moreover, they are not designed for the global, passive, and external adversary model and nor provide unobservability. In our schemes, no on-demand routing is necessary because every sensor knows where to forward its messages.

In the context of sensor networks, techniques [9] for hiding the base station (message destination) from an external global adversary are studied. In their schemes, every sensor node is a mix and also transmits at a constant rate. This scheme however is expensive for sensor networks. In [24, 14], a random walk based phantom flooding scheme is proposed to defend against an external adversary who attempts to trace back to the data source in a sensor network where sensor nodes report sensing data to a fixed base station. A more recent work [30] demonstrates an example attack against the flooding method used in [24, 14] and proposes a new random walk algorithm. In [13], a path confusion algorithm is proposed to increase source location anonymity. Note that these schemes do not provide unobservability and only work for a local adversary model.

To hide real event messages, in a baseline scheme with perfect privacy, every sensors send out periodic messages

following a certain constant or probabilistic interval. However, in this case, real event messages will need to be postponed to send out in the next transmission point, which causes high real event message transmission latency. [28] identified such a fundamental tradeoff between performance and privacy, in which real event messages are transmitted as soon as they can, with disturbed time intervals that could not be detected by available statistical methods. Hence, the real event message transmission latency is reduced and meanwhile statistically strong source anonymity for sensor networks could be achieved. Clearly, this work is complementary to ours and they can be seamlessly integrated to provide both low latency and low communication overhead. In [22], also under the global attacker model, two schemes are proposed. The first one is a periodic collection scheme that could be applied in situations when a certain degree of message overhead and message delivery latency could be tolerated, because all sensors send messages at a low rate; the second one is a k-anonymity like source-simulation scheme, where only  $k - 1$  fake sources simulate the mobility pattern of a mobile real source, so that the communication overhead could be reduced at the cost of sacrificing some privacy.

## 8. CONCLUSION

In this paper, we solve the optimal proxy placement problem by using local search heuristics and propose a Proxy-based Filtering Scheme (PFS) and a Tree-based Filtering Scheme (TFS), which are simple yet efficient event source unobservability preserving solutions for sensor networks. The two methods work together, so that we can maximally reduce the network traffic while increasing the delivery ratio without sacrificing privacy. Performance evaluation demonstrates that our schemes can largely improve the system performance compared with a baseline scheme.

As our future work, we will investigate more on the TFS scheme, for example, how to further optimize buffer time and size for each proxy level. Other attack models such as local and/or insider attacks are also of interest to us.

**Acknowledgement:** We would like to thank the anonymous reviewers for their valuable comments and suggestions.

## 9. REFERENCES

- [1] Csim 19, <http://www.mesquite.com/>.
- [2] Glomosim, <http://pcl.cs.ucla.edu/projects/glomosim/>.
- [3] Mica2 mote, <http://www.xbow.com/>.
- [4] TinyOS. <http://www.tinyos.net>.
- [5] Anonymity bibliography, <http://freehaven.net/anonbib/date.html>, 2005.
- [6] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, 2001.
- [7] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 2000.
- [8] D. Chaum. Untraceable electronic mail, return address, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

- [9] J. Deng, R. Han, and S. Mishra. Intrusion tolerance and anti-traffic analysis strategies for wireless sensor networks. *International Conference on Dependable Systems and Networks (DSN'04)*, June 2004.
- [10] C. Díaz and B. Preneel. Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, 2004.
- [11] S. Fischer-Hubner. *Privacy Enhancing Technologies*, volume 1958. Springer Berlin Heidelberg, 2001.
- [12] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In *Proceedings of Information Hiding: First International Workshop*, 1996.
- [13] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. *Securecomm*, 2005.
- [14] P. Kamat, Y. Zhang, W. Trappe, and C. Ozturk. Enhancing source-location privacy in sensor network routing. In *ICDCS '05*, pages 599–608, 2005.
- [15] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.
- [16] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH 1998)*, 1998.
- [17] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. John Wiley and Sons, Inc., 1975.
- [18] J. Kong and X. Hong. Anodr: anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *MobiHoc '03*, 2003.
- [19] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 1–10, 1998.
- [20] W. Zhang, M. Tran, S. Zhu, and G. Cao. A Compromise-Resilient Scheme for Pairwise Key Establishment in Dynamic Sensor Networks. In *ACM Mobihoc*, 2007.
- [21] D. Liu, P. Ning, and W. Du. Attack-resistant location estimation in sensor networks. In *Proceedings of The 4<sup>th</sup> International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.
- [22] K. Mehta, D. Liu, and M. Wright. Location privacy in sensor networks against a global eavesdropper. In *ICNP*, 2007.
- [23] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.
- [24] C. Ozturk, Y. Zhang, and W. Trappe. Source-location privacy in energy-constrained sensor networks routing. *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'04)*, October 2004.
- [25] R. Cimikowski, E. Mooney. Proximity-based adjacency determination for facility layout. In *Comput. Ind. Eng.*, 32(2):341–349, 1997.
- [26] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, L. Yin, S. Shenker, and F. Yu. Data-centric storage in sensornets. In *Proceedings of ACM First Workshop on Hot Topics in Networks*, 2001.
- [27] M. Shao, S. Zhu, W. Zhang, and G. Cao. pDCS: Security and privacy support for data-centric sensor networks. In *IEEE INFOCOM*, 2007.
- [28] M. Shao, Y. Yang, S. Zhu, and G. Cao. Towards Statistically Strong Source Anonymity for Sensor Networks. In *IEEE INFOCOM*, 2008.
- [29] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.
- [30] Y. Xi, L. Schwiebert, and W. Shi. Preserving source location privacy in monitoring-based wireless sensor networks. In *SSN '06*.
- [31] Y. Zhang, W. Liu, , and W. Lou. Anonymous communications in mobile ad hoc networks. In *IEEE INFOCOM*, 2005.
- [32] B. Zhu, Z. Wan, M. S. Kankanhalli, F. Bao, and R. H. Deng. Anonymous secure routing in mobile ad-hoc networks. In *LCN*, 2004.
- [33] Y. Zhu and R. Bettati. Compromising location privacy in wireless networks using sensors with limited information. In *ICDCS 2007*.