Ad Hoc Networks 11 (2013) 1820–1835

Contents lists available at SciVerse ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

A delay-bounded event-monitoring and adversary-identification protocol in resource-constraint sensor networks ☆

Jinkyu Koo [a,*,1], Dong-Hoon Shin [b,1], Xiaojun Lin [c], Saurabh Bagchi [c]

[a] Samsung Electronics Co., LTD, 416 Maetan 3-dong, Youngtong-gu, Suwon, Gyeonggi-do 442-600, Korea
[b] School of Electrical, Computer and Energy Engineering, Ira A. Fulton School of Engineering, Arizona State University, Office: GWC 450, Tempe, Arizona 85287, USA
[c] School of Electrical and Computer Engineering, Electrical Engineering Building, Purdue University, 465 Northwestern Ave., West Lafayette, IN 47907-2035, USA

ARTICLE INFO

Article history:
Received 28 August 2012
Received in revised form 20 February 2013
Accepted 10 April 2013
Available online 18 April 2013

Keywords:
Security
Event monitoring
Byzantine adversary
Sensor networks

ABSTRACT

Event monitoring is a common application in wireless sensor networks. For event monitoring, a number of sensor nodes are deployed to monitor certain phenomenon. When an event is detected, the sensor nodes report it to a base station (BS), where a network operator can take appropriate action based on the event report. In this paper, we are interested in scenarios where the event must be reported within a time bound to the BS possibly over multiple hops. However, such event reports can be hampered by compromised nodes in the middle that drop, modify, or delay the event report.

To defend against such an attack, we propose SEM, a Secure Event Monitoring protocol against arbitrary malicious attacks by Byzantine adversary nodes. SEM provides the following provable security guarantees. As long as the compromised nodes want to stay undetected, a legitimate sensor node can report an event to the BS within a bounded time. If the compromised nodes prevent the event from being reported to the BS within the bounded time, the BS can identify a pair of nodes that is guaranteSchool of Electrical and Computer Engineeringed to contain at least one compromised node. To the best of our knowledge, no prior work in the literature can provide such guarantees.

SEM is designed to use the minimum level of asymmetric cryptography during normal operation when there is no attack, and use cryptographic primitives more liberally when an attack is detected. This design has the advantage that the overall SEM protocol is lightweight in terms of the computational resources and the network traffic required by the cryptographic operations. We also show an operational example of SEM using TOSSIM simulations.

1. Introduction

Over the past years, wireless sensor networks (WSNs) have received a great amount attention as a promising technology for a variety of applications. One of the application scenarios for WSNs is in the domain of event monitoring systems [1,2]. In event monitoring, the WSN is deployed over a region where some phenomenon is to be monitored. For example, a number of sensor nodes could be deployed over a battlefield to detect enemy intrusion. When the sensor nodes detect the event being monitored, the event is reported to a base station (BS), which can then be used by a network operator to take appropriate actions.

☆ This material is based in part upon work supported by the National Science Foundation under Grant CNS-0831999. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

* Corresponding author.
E-mail addresses: jinkyu.koo@samsung.com (J. Koo), Donghoon.Shin.2@asu.edu (D.-H. Shin), linx@ecn.purdue.edu (X. Lin), sbagchi@purdue.edu (S. Bagchi).

[1] The work by Jinkyu Koo and Dong-Hoon Shin on this paper was done when they were Ph.D. students at Purdue University.

1570-8705/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2013.04.005

Significant work in WSNs to date has focused on making data gathering (equivalently, event monitoring) energy-efficient. In this work, we are concerned with timely and secure reporting of event information, even in the presence of adversarial nodes. The adversarial nodes can include some that have been compromised and therefore possess the cryptographic keys.

We give two examples of application scenarios where accurate and timely collection of sensor data in a multi-hop setting is important. First consider the smart power grid where myriad sensors will be deployed through the transmission and distribution infrastructure. State estimation is performed based on the inputs from the sensors to determine if corrective action is needed, such as, reducing the load on a sub-station. Incorrect state estimation can cause devastating consequences such as blackout of a region [3]. Such incorrect state estimation can be caused by delaying or dropping meter readings from a modest fraction of sensors. A wireless surveillance system using infrared (IR) beam sensors is another example, since many of the commercially-available wireless IR beam sensor nodes are plugged into existing electrical outlets [4]. In this case, missing an event report can lead to failure in taking immediate actions when a burglar attempts a forcible entry. Note that in both of these cases, there is no energy constraint on the network, but there is a high security requirement. Also, in both these cases, physically protecting the nodes is difficult and therefore one has to acknowledge the possibility of nodes, which were originally a part of the network, being compromised.

Although relaxing the energy constraints gives us more freedom to design sophisticated control protocols, it remains a difficult task to secure the event reporting process when the monitoring network is under attack. First, the sensor nodes are inherently vulnerable to attacks because they are usually deployed in non-protected environments. The adversary can often easily access the sensor nodes, and may even compromise them by reprogramming them. This is particularly of concern since efficient reprogramming protocols have been developed [5] and in practice, they rarely use cryptographic security. Once some sensor nodes in a monitoring sensor network are compromised, they may prevent other legitimate sensor nodes from reporting information in a timely manner. Second, the sensor nodes are often based on inexpensive platforms with low computational and communication capabilities. Hence, there are sill significant limitations in terms of the amount of expensive cryptographic mechanisms that can be used. For example, in the Micaz platform, TinyECC public-key cryptography [6] takes about 2 s to generate a single signature and about 2.4 s to verify a signature! As a result, we cannot liberally use such cryptographic mechanisms due to the excessive computation and communication overhead, for fear of delaying the event-report process.

For this reason, we propose Sem, a secure event monitoring protocol that can work even when there exist compromised nodes in the network. We are targeting a multi-hop network scenario where all sensor nodes except the BS node can be compromised. The compromised nodes can launch arbitrary attacks in a Byzantine manner, such as dropping, modifying, and delaying the event report. They may also arbitrarily collude among themselves. Even in such a hostile environment, Sem can provide the following provable security guarantees:

- As long as the compromised nodes want to avoid being detected, a legitimate sensor node can report an event to the BS within a bounded time.
- If the compromised nodes launch an attack that causes the event report not to arrive at the BS within the bounded time, the BS can identify a pair of nodes that is guaranteed to contain at least one compromised node.

We believe that in many practical scenarios, the adversary has the incentive to keep the compromised nodes from being detected, because otherwise, the network operator will be able to remove or reprogram the detected compromised node one by one, eventually defeating the attack. Hence, the above security guarantees are meaningful and useful to attain.

Our design of Sem is parsimonious in its usage of resources - both in the usage of expensive computations for cryptographic operations, and in generating additional network traffic. Specifically, Sem makes a distinction between a normal operation mode and a diagnosis mode (when some attack has been detected and culprit nodes are sought to be identified). In the normal operation mode, Sem only uses a minimal (and indispensable) level of asymmetric cryptography. In detail, the BS signs on a special packet to collect the event reports, and the normal sensor nodes verify only the signature from the BS. This is practically important because we expect that for most networks, the vast majority of the time will be the normal mode of operation, free of security attacks.

The remainder of this paper is organized as follows. Section 2 gives an overview of the previous works for event monitoring. In Section 3, we formally state our objective, assumptions, and notations. In Section 4, we discuss what approach we have to take to achieve our objective and provide a straw-man protocol. In Sections 5 and 6, we present Sem in detail, and show the advantage of Sem over the straw-man protocol. Section 7 discusses the relevant miscellaneous issues. We provide experiment results for Sem in Section 8, and give concluding remarks in Section 9.

2. Related works

Event monitoring applications of WSNs have been studied for a variety of scenarios, including military surveillance and forest-fire detection. A common research issue of event monitoring is energy efficiency and lifetime maximization of sensor networks. Several schemes are proposed to address the optimization of sensing coverage (e.g., [1,7,8]), the goal of which is to monitor the system of interest using the minimal amount of energy resources. Another direction to improve energy efficiency is to balance the energy consumption over the sensor nodes (e.g., [7,9]), since unbalanced energy dissipation causes some nodes to die faster than others, thus reducing the network lifetime. However, most of existing protocols are designed without security in mind. Hence, the compromised nodes

in the middle of the route from the reporting sensor node to the BS can easily break the protocols by dropping/modifying/delaying the event report.

Recently, event monitoring in the presence of compromised nodes began to receive attention [10]. The authors in [10] assume the Man-in-the-middle attackers that can drop, delay, or manipulate the event report from a legitimate sensor node. Their approach to defend against the compromised nodes is to make the sensor nodes flood the event report over the entire network. This method will work when there exists at least one "legitimate route" from each reporting node to the BS that does not contain any compromised nodes. However, if the adversary simply compromises all the neighboring sensor nodes of the BS, and thus isolating the BS, this method cannot provide any guarantees.

Detecting malicious actions by intermediate nodes has been studied in the area of path-quality monitoring [11–14]. In path-quality monitoring, the goal is to reliably raise an alarm when the packet-loss rate and the packet-delay exceed certain thresholds. There are two common approaches. One is to make a destination node return the number of the packets that it receives from a source node. The other is to make the source node perform the active probing (e.g., ping or traceroute) on the routing path. However, just detecting misbehaviors, without identifying the malicious nodes, is insufficient. Consider the case when adversary nodes encircle the BS. All paths are measured to be of low quality. However, there is still no way to carry out communications between a sensor node and the BS. This scenario motivates us to introduce a new secure guarantee of identifying malicious nodes. By pin-pointing the malicious node, the BS can then make the network operator take it away from the network or replace it.

Localizing the malicious node is a more difficult problem than just detecting the malicious activity. Much of the early work tackled this problem through the mechanism of overhearing [15,16]. However, the solutions in this category do not require an evidence when a node reports a misbehavior, and thus cannot handle false misbehavior reporting and collusion among Byzantine adversaries. Subsequent work – ODSBR [17] and PAAI [18] – can handle colluding Byzantine adversaries by using acknowledgment-based approaches that require use of onion-manner signing from nodes between the source and the destination. Since the onion-manner signed acknowledgment mechanism takes a high overhead (too much time, bandwidth, and packet length), ODSBR and PAAI use this to identify a faulty link only after the packet-drop rate becomes higher than a threshold. However, any single event report can be a critical one (imagine the enemy invasion detection in a battle field). Thus, losing the event reports until the threshold test fails is not acceptable from the event-monitoring viewpoint. One could resolve this difficulty by a simple modification to ODSBR and PAAI in such a way that the onion-manner signed acknowledgment is required for every data packet from every node on a traffic route (see the straw-man protocol in Section 4). However, such a modification makes the scheme infeasible for resource-constrained wireless networks because we go back to the issue of high computation and communication overhead.

To the best of our knowledge, no existing solution can work in hostile environments where the compromised nodes may block *all* the routing paths coming into the BS, thereby leaving no chance for a legitimate node to report an event in time. In our work, we focus on resolving this issue and providing provable security guarantees with consideration of communication and computation overhead.

3. Problem statement

We consider a multi-hop wireless sensor network that consists of a base station (BS) node and a number of sensor nodes. A sensor node is in charge of sensing a delay-sensitive event like a power line shutoff. A network operator monitors the sensor network through the BS that attempts to collect the events (if any) from the sensor nodes. It is important that if an event occurs at a sensor node, the BS gets informed of it as soon as possible in order for the network operator to take action in time.

Since some sensor nodes are more than one hop away from the BS, the events sensed at such nodes are reported to the BS through a multi-hop routing path as shown in Fig. 1. However, if a node in the middle of the routing path is compromised, the compromised node may drop/modify the event report, or delay it for a very long time. This problem cannot be resolved even if a legitimate node sends the event by flooding over the entire network, because the BS may be completely encircled by compromised nodes. For example, in Fig. 1, if nodes *a* and *f* are compromised, no other sensor nodes can successfully report an event to the BS even by flooding. Isolating BS is often easy to achieve because the neighboring nodes of BS are not that many.

3.1. Objective

The objective of this paper is to overcome the aforementioned difficulty in monitoring delay-sensitive events. We want to devise a protocol that provides the following provable security guarantees. (1) As long as the compromised nodes want to stay undetected, a legitimate node can report an event to the BS within P time units; and (2) If the compromised nodes launch an attack that causes the event report from a legitimate node not to reach the BS within P time units, the BS can identify a pair of nodes that is guaranteed to contain at least one compromised node.

Note that with the above security guarantees, we can defeat an attack even in the situation where the BS is com-

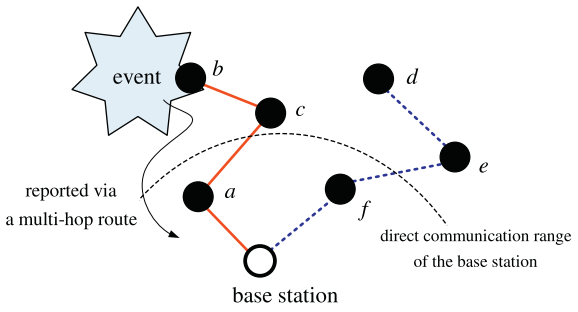


Fig. 1. Multi-hop routing paths to collect events.

pletely encircled by compromised nodes. This is because the network operator will be able to remove or reprogram the detected compromised node one by one, eventually securing a safe route to the BS for collecting event reports.

Note also that we do not guarantee that all sensor nodes will report a detected event to the BS within P time units: once a node is compromised, the event occurred at the compromised node may not be reported. Intuitively, if the reporting node is itself compromised, there is no way to get informed of the event that occurs at the node. However, if an event is sensed by multiple sensor nodes, say n sensors, and at least one of them is legitimate, we can still guarantee that the event is reported to the BS within P time units by our protocol. To make our protocol useless, the adversary needs to compromise all n sensor nodes. In many practical scenarios, the value of n may be large (e.g., all sensor nodes around an invasion route by the enemy movement). In contrast, without our protocol, the adversary only needs to compromise the neighboring nodes of the BS to nullify the monitoring system. Thus, the security benefit from our protocol is significant.

3.2. Assumptions

(1) We consider a stationary network: the locations of all nodes are fixed. (2) Only the BS can be trusted, i.e., any sensor node can be compromised and then behave maliciously. The compromised nodes may arbitrarily collude among themselves. (3) Byzantine adversary model is considered, i.e., the compromised nodes can take arbitrary malicious actions. For example, the compromised node may drop, modify, or delay the event reports. Further, they may launch a jamming attack to prevent some nodes in the network from communicating. However, for ease of presentation, we assume for the time being that there is no jamming attack. We will relax this assumption in Section 7. (4) All links are bi-directional. (5) We assume that transient packet losses (e.g., due to temporary bad channel quality) can be recovered by a lower-layer automatic repeat request (ARQ) mechanism. Thus, we assume that nodes do not fail unless they are compromised. We will also relax these assumptions in Section 7. (6) The time to transmit a packet across one hop, including retransmissions by ARQ, is bounded above by B time units. (7) A node shares a secret key with each of its *neighboring nodes*. We assume that when a node sends any packet to its neighboring node, it sends a secret-key-encrypted message together. By this, any node can authenticate neighboring sender's identification (ID).[2] (8) The public key of a node is assumed to be known to every other node. How to manage the public key infrastructure is out of the scope of this paper. (9) Events monitored by the network occur infrequently.

[2] In the proposed protocol presented in Section 5, a node is required to receive a packet from a designated sender. Thus, authenticating the sender's ID is important, because otherwise an arbitrary neighboring node of the receiver can maliciously claims to be the designated sender. We note that since a node is sharing a secret key only with its neighboring nodes, the overhead to maintain the secret key is much lower than in the case where a node shares a secret key with all other nodes in the network.

3.3. Notations and definitions

(1) $[X_1, \ldots, X_n]$ returns the concatenation of the input strings $X_1, \ldots, X_n$. (2) We use $N(a)$ to denote 'node a' in short. (3) $S_a(X)$ returns a signed message for the input string X made by $N(a)$. The signed message is defined as $S_a(X) = [X, \mathrm{H}(X), \mathrm{SIG}_a(\mathrm{H}(X))]$, where $\mathrm{H}(X)$ is a hash of the input string X, and $\mathrm{SIG}_a(x)$ is a signature created on the string x by $N(a)$ using its private key. (4) $ID(a)$ denotes the identification (ID) of $N(a)$. (5) $SET(a_1, \ldots, a_n)$ denotes the set of nodes $N(a_1), \ldots, N(a_n)$. (6) A suspicious set is a set of nodes that includes at least one compromised node in it.

4. Straw-man protocol

In this section, we first discuss what are the basic methodologies needed for detecting a malicious activity, and why just detecting a malicious activity is not good enough for identifying the malicious node. Then, based on such a discussion, we provide a straw-man solution to achieve our objective. However, this straw-man solution incurs a high computation and communication overhead. This problem will motivate our proposed protocol in the next section.

4.1. What do we need?

First, as we have seen in Section 3, if we simply let the sensor nodes send a report to the BS only when they detect an event, the compromised node in the middle of a route may drop this event report, or hold it for a long time. Thus, the BS may not know within a bounded time if the event has occurred. To detect this type of attacks, we therefore need periodic reporting, i.e., a sensor node is required to periodically report something to the BS, whether or not it detects an event. In this way, if the BS does not get a new report from a certain node within a time bound, the BS can recognize that an attack is in effect. Second, in order to detect modification to the event reports, we will also need to use some signature scheme, e.g., through public-key cryptography.

However, we caution that the use of periodic reporting and public-key cryptography alone will not provide a satisfactory solution to our problem. First, letting every sensor node send a signed report periodically, regardless of whether it has an event to report, leads to significant computation and communication overhead. Second, even with such a large overhead, these mechanisms alone do not guarantee that the BS can always identify who the malicious node is after detecting a malicious activity. This is because the BS has to rely on other nodes' opinions to locate the culprit, but the BS cannot trust anybody except itself. Further, the malicious may start acting normally as soon as the BS detects a malicious activity. Thus, the BS may not be able to find enough evidence to convict the malicious node.

Therefore, what we need for achieving our objective is a protocol that uses the aforementioned methodologies to detect a malicious behavior and collects incriminating evidence for the malicious behavior at the same time. We note that we can make such a protocol by adopting the onion-manner acknowledgment (ACK) mechanism with staggered timeout, which is used in ODSBR [17]. We refer to this protocol as straw-man protocol, and introduce it below as our baseline protocol.

4.2. Detail of the straw-man protocol

4.2.1. A route to collect event reports

The straw-man protocol organizes sensor nodes in multiple line networks that originate from the BS. For example, in Fig. 1, the BS, $N(a)$, $N(b)$, and $N(c)$ form a line network, while $N(d)$, $N(e)$, $N(f)$, and the BS form another line network. In the straw-man protocol, the BS collects events from each line network separately. Thus, without loss of generality, we consider one line network model of K hops as shown in Fig. 2. In this model, we denote by $N(0)$ the BS, and by $N(i)$ the sensor node at a i-hop distance from the BS. Define $I = \{0, 1, \ldots, K\}$. We refer to the direction from $N(0)$ to $N(K)$ as the forward direction, and the opposite as the backward direction.

4.2.2. Malicious activity detection

Algorithm 1. Straw-man protocol at $N(i)$

```
1:  if N(i) = N(0) then
2:    transmit a PT to N(1) every P/2 (or less) time units
3:    start a timer t_a(0)
4:    if receiving an ACK before the timer t_a(0) expires then
5:      if the ACK contains event reports then
6:        deal with the event reports
7:      end if
8:      if the ACK is from N(i) ≠ N(K) then
9:        investigate malicious nodes (refer to Section 4.2.5)
10:     end if
11:   else
12:     go to line 9
13:   end if
14: else if N(i) = N(K)
15:   if receiving a PT then
16:     generate an ACK and send it to N(K − 1)
17:   end if
18: else
19:   if receiving a PT then
20:     forward the PT to N(i + 1)
21:     start a timer t_a(i)
22:   end if
23:   if receiving an ACK before the timer t_a(i) expires then
24:     sign on the ACK and forward it to N(i − 1)
25:   else
26:     generate an ACK and send it to N(i − 1)
27:   end if
28: end if
```

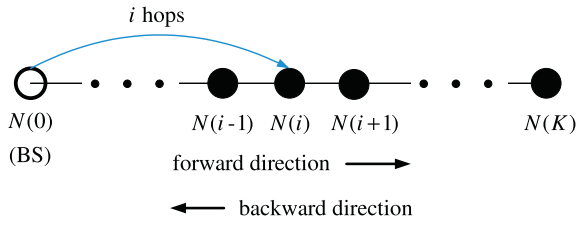


Fig. 2. Line network model to collect event reports.

The pseudo-code of the straw-man protocol is given in Algorithm 1. In straw-man protocol, $N(0)$ (i.e., BS) sends out a probe token (PT) in the forward direction through the line network (line 2). The PT is defined as

$$\mathrm{PT} = S_0([ID(0), R, Q]), \tag{1}$$

where R denotes the routing information of the line network model in Fig. 2, and Q is an integer variable, called round sequence. Every time $N(0)$ sends out the PT, it increases the value of Q by 1 to prevent the old PT from being re-used in replay attacks [19]. When $N(K)$ receives the PT, it generates an ACK packet (line 16). On receiving the ACK, $N(i)$ for any $i \in I\backslash\{0,K\}$ forwards the ACK in the backward direction (line 24). If $N(i)$ has an event to report, it can embed the report into the ACK. Hence, if $N(0)$ receives the ACK from $N(K)$, then no packet drop has occurred, and $N(0)$ can be informed of any embedded event. On the other hand, $N(0)$ detects a malicious activity when it does not receives the ACK from $N(K)$ (until a threshold time). It is easy to see that if each round is completed within $P/2$ time units, $N(i)$ for any $i \in I\backslash\{0\}$ can report a sensed event to $N(0)$ within P time units.

4.2.3. Timeout setup to identify a malicious node

To identify a malicious node that drops/delays either the PT or the ACK, $N(i)$ for any $i \in I\backslash\{K\}$ starts an ACK timer $t_a(i)$ immediately after transmitting the PT, whose timeout value is set to $B_a(i)$ (line 21). If the ACK has not been received from $N(i+1)$ before $t_a(i)$ goes off, $N(i)$ gives up waiting and generates its own ACK packet (line 26). Ideally, only when $N(i+1)$ is a compromised node that drops either the PT or the ACK, or delays it beyond a certain limit, $N(0)$ receives the ACK generated by $N(i)$. To ensure this property, if $N(i+1)$ is legitimate, $N(i+1)$ should always be able to send the ACK to $N(i)$ before $t_a(i)$ expires. Due to this reason, the nodes calculate the maximum time required for the PT to traverse from themselves to $N(K)$ and for the ACK to return to the current position along the reverse route. Then, $N(i)$ for any $i \in I\backslash\{K\}$ sets the timeout $B_a(i)$ to this maximum value, which is equal to

$$B_a(i) = (K - i - 1)B + (K - i)B. \tag{2}$$

Note that $B_a(i)$ is larger than $B_a(i+1)$ by $2B$. As Fig. 3 illustrates, this allows $N(i+1)$ to send $N(i)$ the ACK before $t_a(i)$ expires. This ACK can be either the one that $N(i+1)$ has received from $N(i+2)$, or the one that $N(i+1)$ generates on its own. More formally, we can state this property as the following lemma.

Lemma 1. *For any $i \in I\backslash\{K\}$, if both $N(i)$ and $N(i+1)$ are legitimate, $N(i)$ has no reason to generate its own ACK.*

Proof. Remember that nodes start the ACK timer right after transmitting the PT. Since $N(i+1)$ holds the PT for B time units at most, $t_a(i+1)$ starts within B time units after $t_a(i)$ started. This implies that $t_a(i+1)$ goes off at least B time units earlier than $t_a(i)$ does, since $B_a(i) = B_a(i+1) + 2B$. Thus, $N(i+1)$ can alway send an ACK to $N(i)$, whether it is what $N(i+1)$ generates on its own when $t_a(i+1)$ expires, or it is what $N(i+1)$ has received from $N(i+2)$ before $t_a(i+1)$ expires. Hence, $N(i)$ should not be the node that generates its own ACK. □

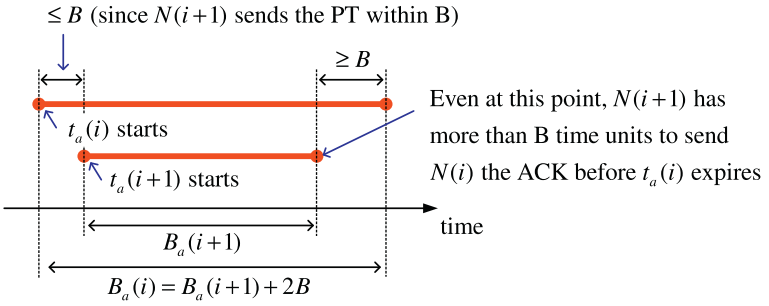


Fig. 3. Staggered timeout.

4.2.4. Event report in backward direction

To prevent the compromised nodes from modifying the ACK (thereby falsely accusing legitimate nodes), or tampering with an event report in the ACK, the nodes are required to sign on the ACK. In detail, the ACK generated by $N(i)$ is defined as

$$\text{ACK} = S_i([ID(i), Q, E]), \tag{3}$$

where E denotes the event report. If nodes do not have anything to report, they leave E as E = NULL. When $N(i)$ forwards the ACK from $N(i+1)$ to $N(i-1)$ (i.e., it does not generate its own ACK), the ACK sent by $N(i)$, denoted by ACK (i), is defined as

$$\text{ACK}(i) = S_i([\text{ACK}(i+1), ID(i), E]). \tag{4}$$

Note that the ACK is signed by the nodes in an onion manner, i.e., the ACK sent by a node encapsulates the ACK (if any) sent by higher-indexed nodes. Thus, a compromised node cannot modify or forge the ACK from any higher-indexed node, unless they collude with each other.

4.2.5. Investigation of malicious nodes

On receiving the ACK, the BS can investigate the malicious nodes (line 9), providing the following guarantee.

Proposition 1. *If N(0) gets the ACK generated by N(i) for $i \neq K$, SET(i, i + 1) is a suspicious set.*

Proof. Assume that $SET(i, i+1)$ is not a suspicious set. Then, $N(i)$ and $N(i+1)$ are both legitimate. Since no compromised node can forge the signature of a legitimate node, the ACK that $N(0)$ has received is indeed generated by $N(i)$. This implies that $N(i)$ has received the PT, and thus forwarded the PT to $N(i+1)$. Hence, by Lemma 1, $N(i)$ cannot be the node that generate its own ACK. This is a contradiction. □

Note that a suspicious set contains at least one compromised node, which is either the one that dropped or delayed the PT or ACK, or one of its colluding partners. In addition to what Proposition 1 guarantees, $N(0)$ can also identify the node that modified the content of the ACK (if any), by sequentially verifying the signatures on the ACK from the outer shell. That is, if there exists $N(j)$ such that the signature of $N(j)$ is valid, but the signature of $N(j+1)$ is invalid, $SET(j, j+1)$ is a suspicious set [17,18].

4.2.6. Issue in the straw-man protocol

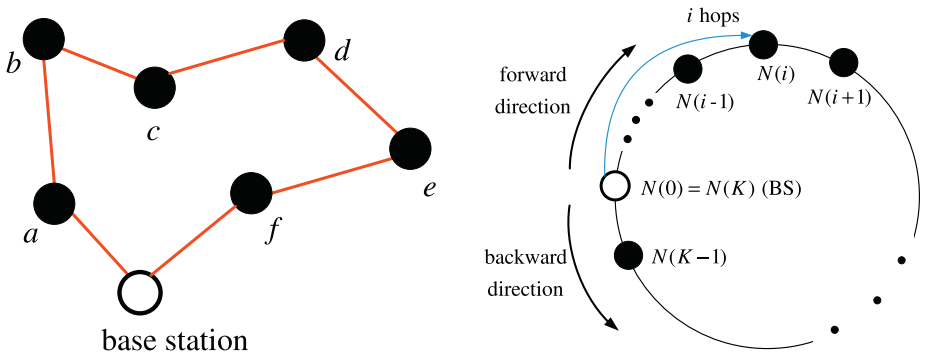
We can see that the straw-man protocol achieves our objective: it can collect the event reports from sensor nodes within P time units if there is no attack; Otherwise, it can identify a suspicious set that contains at least one compromised node. However, the straw-man protocol always requires the sensor nodes to send a signed packet as in (4), i.e., the straw-man protocol needs to use an expensive ACK mechanism signed in an onion manner every $P/2$ time units, whether or not an event occurs. This is important for providing the guarantee of identifying a suspicious set whenever an event is not received on time. Since the occurrence of events is unpredictable and the PT generation by the BS has to be periodically done, the sensor nodes are required to sign on the ACK in an onion manner even though they may not have anything to report and even when there is no attack going on (the normal case). Clearly, this is expensive since the onion-manner signing technique causes a heavy computation overhead for relaying the ACK. Further, it results in a large payload size for the ACK that often needs to be fragmented into multiple packets.

5. Proposed protocol: SEM

The issue with the straw-man protocol is that it requires an expensive onion-manner signed ACK mechanism even in normal operations. Ideally, we would like to design a protocol that has low overhead in normal scenarios, i.e., when there is no attack, and only invokes the heavier-overhead mechanism when an attack is suspected. For this purpose, we now propose our new protocol, called Sem (Secure Event Monitoring).

5.1. Overview

At a high level, Sem preserves a similar structure to the straw-man protocol: the BS node periodically sends out the PT through the line network of Fig. 2, and gets an ACK that identifies the malicious link (if any) based on the staggered timeout mechanism. However, in order to reduce the overhead, Sem first detects if something bad happens, i.e., if there exists a compromised node that hinders the event collection operation. If the event collection operation looks fine, then Sem cancels the ACK timers at the sensor nodes. Thus, Sem uses the expensive onion-manner signed ACK mechanism only when some malicious activity that disturbs the BS's event collection process is detected. By this design, Sem can provide the same security guarantee as the straw-man protocol, while eliminating



(a) An example of the EGC (b) Abstract model of the EGC

Fig. 4. Event gathering circle (EGC).

the heavy communication and computation overhead to send the ACK under normal legitimate scenarios.

Towards this end, SEM forms the line network to collect the event reports in such a way that both the start and the end of the line are the BS (Section 5.2.1). This means that the sensor nodes are physically organized in a circle that passes through the BS. The timeout value at each node is extended in order to cancel the ACK timer when nothing is bad, ensuring to identify the malicious node otherwise (Section 5.2.3). Further, the event report is embedded in the PT, not the ACK, since the ACK is only required when an attack is detected (Sections 5.2.2 and 5.2.4). The detailed description of SEM is provided in the following.

5.2. Detail of SEM

5.2.1. A circular route to collect event reports

SEM organizes the nodes into a circular network that passes through the BS as shown in Fig. 4a. We refer to the circular route as event gathering circle (EGC). If the BS finds a situation that it cannot include all the sensor nodes in the network into one single EGC, e.g., due to scalability considerations, the BS can form multiple EGCs, and collect events from each EGC independently. As long as each sensor node belongs to at least one EGC, the BS can collect the event reports from the entire network. Thus, without loss of generality, we consider the case that there is only one EGC in the network that is modeled as a circular route of K hops as in Fig. 4b. Note that in this model, $N(i)$ denotes the sensor node at a i-hop distance from the BS in the forward direction, thus implying that both $N(0)$ and $N(K)$ represent the BS.[3] We can think of the network in Fig. 4b as the network in Fig. 2 where $N(0) = N(K)$.

[3] A sensor node can appear in an EGC more than once, although it is not recommended due to its inefficiency. Thus, a preferred network topology would be the one where we can form EGCs in such a way that a sensor node belongs to a unique EGC only once. However, in some topology (e.g., a tree), one may be forced to use the same node more than once an EGC. If a compromised node acts as for example, $N(i)$ and $N(j)$ $(i \neq j)$ in the same EGC, then $N(i)$ and $N(j)$ can be considered as equivalent to collaborating malicious nodes.

5.2.2. Detection of malicious activity

Algorithm 2. SEM at $N(i)$

```
1:  Notation: X@Q denotes that an object X is for round Q
2:  if N(i) = N(0) or N(i) = N(K) then
3:    if receiving a PT@Q from N(K − 1)
4:      if the PT@Q contains event reports
5:        deal with the event reports
6:      end if
7:      if t_a(0)@Q < T_th then
8:        tansmit a PT@(Q + 1) to N(1) when t_a(0)@Q = T_th
9:        start t_a(0)@(Q + 1)
10:     else
11:       wait for T_m time units
12:       generate an ACK@Q and send it to N(K − 1)
13:     end if
14:   end if
15:   if receiving an ACK@Q before t_a(0)@Q expires then
16:     if the ACK@Q is from N(i) ≠ N(K) then
17:       investigate malicious nodes (refer to Section 4.2.5)
18:     end if
19:   else
20:     go to line 17
21:   end if
22: else
23:   if receiving a PT@Q then
24:     cancel t_a(i)@(Q − 1)
25:     forward the PT@Q to N(i + 1)
26:     start t_a(i)@Q
27:   end if
28:   if receiving an ACK@Q before t_a(i)@Q expires then
29:     sign on the ACK@Q and forward it to N(i − 1)
30:   else
31:     generate an ACK@Q and send it to N(i − 1)
32:   end if
33: end if
```

Algorithm 2 gives the pseudo-code of SEM. The basic framework of SEM for sending PT and ACK is almost the same as the straw-man protocol. Namely, $N(0)$ sends out the PT in the forward direction of the EGC in the period of $P/2$ time units or less (line 8). When $N(K)$ receives the PT, $N(K)$ generates a signed ACK packet (line 12) and forward it in the backward direction, in order to identify malicious nodes. Nodes set up the timeout of the ACK timer similarly to (2) (line 26). If $N(i)$ for any $i \in I\backslash\{0,K\}$ receives an ACK from $N(i+1)$ within the timeout, $N(i)$ signs on the ACK and forwards it to $N(i-1)$ (line 29); Otherwise, $N(i)$ generates its own ACK (line 31).

However, in SEM, if a sensor node $N(i)$ has an event to report, it embeds the event report into the PT instead of the ACK as

$$\mathrm{PT}(i) = S_i([\mathrm{PT}(i-1), ID(i), E]), \tag{5}$$

where PT (i) denotes the PT sent by $N(i)$. Here, $N(i)$ embeds the event reports into the PT only when it has an event to report.[4] Since the event report is carried by the PT, the compromised node may want to drop or delay the PT to prevent an event from being reported in time to $N(K)$, which is also the BS. However, note that since $N(0) = N(K)$, the BS node $N(K)$ can measure the circulation time t_c that is defined as the elapsed time of the ACK timer $t_a(0)$ when $N(K)$ receives the PT (recall that $t_a(0)$ starts immediately after $N(0)$ sends the PT out). If all the sensor nodes in the EGC are legitimate, the circulation time t_c should be less than or equal to $(K-1)B$; Otherwise, it is certain that there exists at least one compromised node in the EGC. However, even in such a case, as long as the value of t_c is no larger than some threshold time, say T_{th}, which is less than $P/2$, the BS may skip identifying the compromised node because the BS can still collect the event within P time units after the event occurred (which is our design objective). Therefore, as long as $t_c \leqslant T_{th}(<P/2)$, the BS does not generate an ACK and it cancels the ACK timers at the sensor nodes, through which the BS prevents the nodes from doing the expensive ACK processing. This implies that only when $t_c > T_{th}$, $N(0)$ receives the ACK from the sensor nodes, by which $N(0)$ can identify the malicious node (line 17).

5.2.3. Timeout setup to identify a malicious node and the potential difficulties

In order to give enough time for the BS to cancel the ACK timers at the sensor nodes when $t_c \leqslant T_{th}$, we extend the timeout set-up in (2) by a time margin T_m. Let $B'_a(i)$ be the new timeout of the ACK timer $t_a(i)$ (line 26). Then, we can express $B'_a(i)$ as

$$B'_a(i) = B_a(i) + T_m. \tag{6}$$

In addition, when $N(K)$ is required to send an ACK in the backward direction, we let $N(K)$ start to send the ACK T_m time units after $N(K)$ receives the PT (line 11). Compared to the straw-man protocol, this in effect increases the timeout at every node in Fig. 3 by the same length T_m. Thus, it is easy to see that even after such a change, Lemma 1 still holds, and thus Proposition 1 also still holds. Thus, we have the following.

Corollary 1. *SEM satisfies the same property as the straw-man protocol that if N(0) gets the ACK generated by N(i) for $i \neq K$, SET(i, i + 1) is a suspicious set.*

In order to cancel the ACK timers at the sensor nodes, $N(0)$ simply starts a new round. In detail, if $t_c \leqslant T_{th}$, the BS node (as $N(K)$) does not send an ACK. Instead, the BS (as $N(0)$) sends out the PT of the new round *when the value of the ACK timer $t_a(0)$ in the current round reaches T_{th}* (line 8). This new round's PT acknowledges the old round's PT, and thus the sensor nodes can safely cancel their old ACK timer when receiving the new round's PT (line 24). This leads the first potential difficulty. **Difficulty 1**: here, we should ensure that the value of T_m is long enough so that the new round's PT can arrive at the sensor nodes before their old ACK timer expires.

If $t_c > T_{th}$, the BS node $N(0)$ stops circulating the PT, i.e. $N(0)$ does not try to cancel the ACK timers at the sensor nodes (lines 10–13). Thus, $N(0)$ will receive the ACK signed by the sensor nodes in the onion manner. However, a second potential difficulty arises. **Difficulty 2**: a wrong choice for the value of T_{th} may lead to a situation that the BS cannot identify a suspicious set even after receiving the ACK. To see this, consider the following example (see Fig. 5), where we set the threshold T_{th} as $T_{th} = (K-1)B + B$, i.e., the BS will investigate the EGC using the ACK mechanism if the circulation time t_c is larger by a margin B over the legitimate maximum bound, $(K-1)B$ time units. Suppose that $N(K-2)$ is a compromised node, and it holds the PT for $2.1B$ time units, while other nodes consume B time units to send the PT. Then, $N(K)$ will find $t_c > T_{th}$, and thus generate an ACK after T_m time units, without initiating a new round. However, to deliver the ACK, nodes may need less than B time units, depending on its current computation load and the channel condition. Suppose that every node takes just $0.1B$ time units to send the ACK. In this situation, it is easy to check that $N(K-2)$ can forward $N(K-3)$ the ACK generated by $N(K)$ before the ACK timer of $N(K-3)$ expires. Fig. 5 illustrates this situation. Therefore, the BS cannot find anything wrong with the compromised node $N(K-2)$, since it will get the ACK signed by all nodes in the EGC including $N(K)$. Namely, although $N(K-2)$ holds the PT more than B time units, and the BS detects that $t_c > T_{th}$, $N(K-2)$ can avoid being identified.

Therefore, we now have to answer the following two questions:

- What should be the appropriate value for the threshold T_{th} such that if $t_c > T_{th}$, the BS can always identify a suspicious set?
- What should be the appropriate value for the time margin T_m by which the BS can cancel the ACK timers at the sensor nodes before they go off?

Note that if we can successfully answer these two questions, SEM can achieve our objective, and only use the expensive onion-manner signed ACK mechanism when we need to identify the compromised nodes.

[4] This procedure introduces the vulnerability that a compromised node may simply remove the event report, since the BS has no way to know whether there exists an event report before receiving it. We explain how SEM deals with this threat at Section 5.2.4.

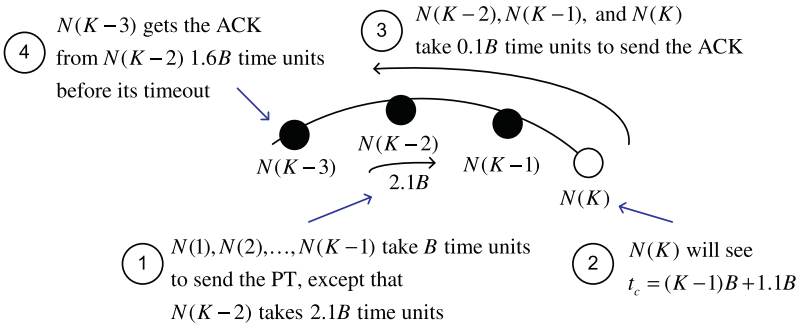


Fig. 5. An example to show that a compromised node may hold the PT for more than B time units without being identified.

In Proposition 2, we will answer the first question, i.e., to determine what is the smallest value of T_{th} that we can set such that the BS is always able to identify a suspicious set when $t_c > T_{th}$. Note that as T_{th} increases, the time required for collecting event reports from a node to the BS goes up linearly (the exact relation is given in Corollary 2) and thus it is beneficial to use small T_{th}.

Proposition 2. *If we set the threshold $T_{th} = B_a(0)$, where $B_a(0) = (2K-1)B$ (obtained by setting $i = 0$ in (2)), the BS can always identify a suspicious set when $t_c > T_{th}$.*

Proof. Suppose that $N(0)$ receives the PT after $t_c > T_{th}$ as depicted in Fig. 6. Then, the time left in the ACK timer of $N(0)$ is $B'_a(0) - t_c = B_a(0) + T_m - t_c < B_a(0) + T_m - T_{th} = T_m$. Namely, the time left in the ACK timer of $N(0)$ is less than T_m. However, as mentioned below (6), $N(K)$ starts to send its own ACK only T_m time units after it receives the PT. Therefore, $N(0)$ will not receive the ACK generated by $N(K)$. Instead, when $N(0)$'s timer expires, $N(0)$ will receive an the ACK that is generated by $N(j)$ for $j \neq K$. In that case, by Proposition 1, $SET(j, j+1)$ is a suspicious set. □

The goal of Proposition 3 is to answer the second of the two questions raised above. Proposition 3 gives us a lower bound to setting the time margin T_m. A larger value of T_m will mean, in the case of an attack, a longer time for a node to generate an ACK in the backward direction. This would mean a longer time for the BS to get the ACK and therefore to identify the suspicious set. Consequently, we would like to set T_m to be as small as possible.

Proposition 3. *If we set the time margin T_m as $T_m \geqslant 2T_{th}$, then no legitimate sensor node will generate its own ACK when $t_c \leqslant T_{th}$ for every round.*

Proof. We prove this by focusing on a typical round Q. It is sufficient to show that the earliest time at which the ACK timer of any node expires is *after* the latest time at which the ACK timer is canceled at that node. Recall that the ACK timer at a node is canceled by the PT reaching the node in the next round.

Consider the situation in Fig. 7, where the BS starts its ACK timer of the Qth round at p_1, and it gets back the PT of the Qth round within T_{th} time units. Then, the BS wants to cancel all nodes' ACK timers that were set up in the Qth round. Thus, the BS starts sending the PT of the $(Q+1)$th round at $p_2 = p_1 + T_{th}$. The last node to receive the PT of the $(Q+1)$th round will be $N(K-1)$. Let $p_3 = p_1 + B + 2T_{th}$. Then, p_3 is the latest time at which the PT of the $(Q+1)$th round reaches back to the BS, because $t_c \leqslant T_{th}$ for both the Qth and the $(Q+1)$th rounds. Here, the extra term B in p_3 is due to the fact that the BS can take up to B time units to transmit the PT of the $(Q+1)$th round. Therefore, the node $N(K-1)$ will receive the PT of the $(Q+1)$th round (and will consequently cancel its ACK timer of the Qth round) before p_3.

Now, consider the time when the ACK timers of the nodes are scheduled to expire. Recall that we are analyzing the earliest possible time when the ACK timers can expire. The earliest time that the ACK timers of any nodes start is p_1. The shortest timeout value is at $N(K-1)$, $B + T_m$ by (2) and (6). Hence, $p_4 = p_1 + B + T_m$ is the earliest time that any timer can expire. This means that the ACK timers for all the nodes will expire at p_4 or later. Now, since $T_m \geqslant 2T_{th}$, we have $p_3 \leqslant p_4$. □

Note that in the proof of Proposition 3, if the BS cannot get the PT of the $(Q+1)$th round before p_3, some legitimate sensor node may generate its own ACK for the PT of the Qth round. However, in this case, the ACK can be discarded by the nodes who have already received the PT of the $(Q+1)$th round, because they know that they have to send the ACK for the PT of the $(Q+1)$th round. Further, in this case, since the BS is expecting to receive an ACK, the statement of Proposition 3 sill holds.

Corollary 2. *If $t_c \leqslant T_{th}$ in every round, which means that the compromised nodes (if any) stay undetected, a legitimate node can report an event within P time units, where $P = 2(B + T_{th})$.*

Proof. By Proposition 2, the BS can finish each round within $B + T_{th}$ time units when no attack is detected (here, the term B comes from the time that the BS takes to send the PT). Thus, each legitimate node in the EGC can report an event within $2(B + T_{th})$ time units, i.e., $P = 2(B + T_{th})$. The factor of 2 comes in because in the worst case, the event report may be generated by a node right after the PT has crossed that node. □

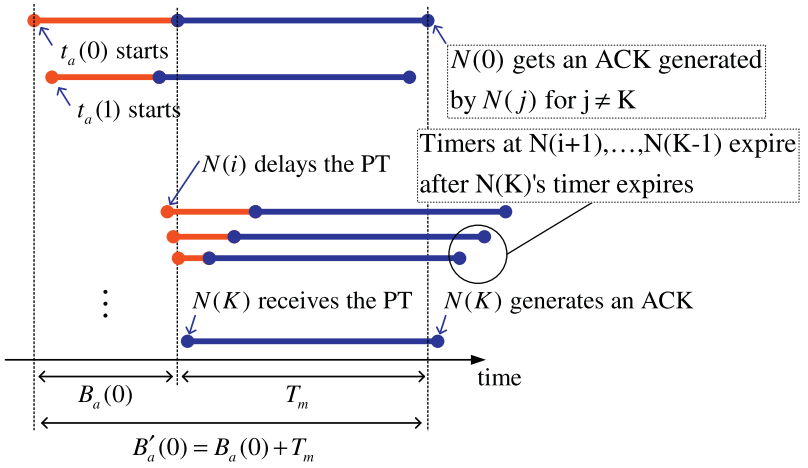


Fig. 6. An example where $N(0)$ receives the PT after $t_c > T_{th}$ because $N(i)$ delayed the PT. Because $N(K)$ generates its own ACK only after $N(0)$ timer expires, the ACK that arrives at $N(0)$ does not contain the ACK from $N(K)$.

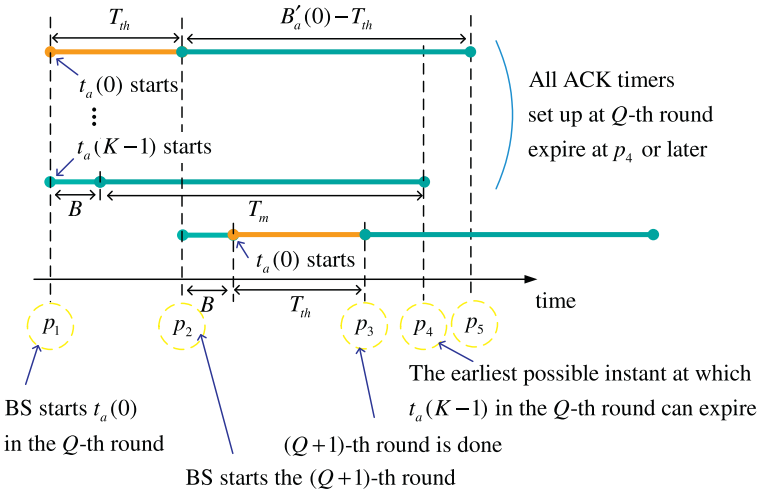


Fig. 7. Time margin T_m. The earliest time instant at which any ACK timer of the Qth round expires (which is p_4) should be later than the latest time instant at which any ACK timer of the Qth round is canceled (which is p_3).

5.2.4. Event report in forward direction

Suppose that a normal node, say $N(i)$, sends an event report by piggybacking it as in (5). A compromised node between $N(i)$ and $N(K)$ may modify the event report, or strip $N(i)$'s signature and event report from the PT leaving only the PT originated by $N(0)$. These attacks cause the BS to get a corrupted event report, or to remain unaware of the event report. To prevent these attacks, when the BS receives event reports, it puts the list L into the PT of a new round, which contains the IDs of nodes who sent the event reports with an authentic signature in the previous round. Thus, $N(i)$ who sent an event report in the previous round should see its ID on the list L in the new round. Otherwise, it implies that somebody after $N(i)$ modified or removed the event report from $N(i)$.

If $N(i)$ does not see its ID on L, it drops the PT and generates an ACK with a piggyback error (PE) message. When the BS receives a PE message, the BS may request the nodes on the EGC to send, in onion-signing manner, the hash value and the signature that $N(i)$ created for the event report, to prove that they have seen the event report from $N(i)$. Based on the collected information from this request, the BS can find at least one suspicious set that contains the highest-indexed node among those who provide the valid hash and signature from $N(i)$, and its next-hop node. We omit the proof since it can be shown in a similar way as Proposition 1. Note that with this approach, we have eliminated the need for all the nodes to sign the event report in an onion manner: only the nodes that have an event to report need to sign their report. Generally, the number of nodes that fall in this category is smaller than the total number of nodes. Hence, this design eliminates unnecessary computational and packet overhead associated with signing in an onion manner.

On the other hand, in order to identify the compromised nodes that modify any contents in the PT, the sensor nodes

verify the signature of $N(0)$ in the PT. If a node $N(i)$ finds $N(0)$'s signature invalid, it sends an event report as E = signature error (SE) in the PT. Then, when the BS gets back the PT, we can guarantee the following.

Proposition 4. *If the BS receives an event report E = SE from $N(i)$ for any $i \in I\backslash\{0\}$, $SET(i-1,i)$ is a suspicious set.*

Proof. Assume $SET(i-1,i)$ is not a suspicious set. Then, both $N(i-1)$ and $N(i)$ are legitimate. Since $N(i-1)$ did not generate the event report E = SE, it must have received authentic signature of $N(0)$ in the PT. Thus, the signatures in the PT that $N(i)$ has received should also be authentic, which means that $N(i)$ cannot be the node that generates the event report as E = SE. This is a contradiction. □

6. Analysis: Advantage of SEM over the straw-man protocol

To see the advantage of Sem, we now calculate how much overhead Sem can reduce from the straw-man protocol, in terms of the number of packet transmissions required.

The main difference in the overhead comes from the fact that in normal operations, Sem does not need the ACK signed in the onion manner. For this reason, we first calculate the payload size of the ACK and thus the number of packets that we need for accommodating the payload.

Consider the case when an attack is detected. When a node sends an ACK, it needs to generate its own signature. The size of the signature is 22 bytes when we use TinyECC [6], which is a popular public key cryptography package for sensor platforms. Since the ACK also includes the sender ID, which is usually 2 bytes, each node needs at least 24 bytes to send an ACK. This means that when the ACK is conveyed from $N(K)$ to $N(0)$ through the EGC, the payload size increases by 24 bytes at each node. However, most of the commercial sensor nodes use IEEE 802.15.4 radios, where the payload size of a packet must be less than 114 bytes. Therefore, a single packet can accommodate the ACK signed by up to four different nodes; If $4m$ nodes sign on an ACK, where m is an integer, the ACK payload should be fragmented to m packets at a sender and reassembled at a receiver. Thus, it is easy to see that if K is a multiple of 4, the total number of packet transmissions for $N(0)$ to receive the ACK from $N(K)$ via the EGC can be expressed as $K(K+4)/8$, i.e., $O(K^2)$. Recall that in the straw-man protocol, the BS has to receive the ACK, whether or not there exists a malicious activity. Thus, each round needs $O(K^2)$ packet transmissions for the ACK. However, Sem does not require the BS to receive the ACK in normal operations. Hence, considering the ACK only, Sem can reduce $O(K^2)$ packet transmission overhead from the straw-man protocol in the normal mode, which corresponds to significant savings.

On the other hand, in both the straw-man protocol and Sem, the PT can be sent in a single packet so that for each round, we need $O(K)$ packet transmissions for the PT circulation. Thus, when there exists no event to report in the normal mode, Sem needs only $O(K)$ packet transmissions in total in each round, while the straw-man protocol needs $O(K^2)+O(K)$ packet transmissions. In Sem, if some node needs to report an event, it signs on the PT as in (5). If there is only one node that needs to report an event, he signs on the PT but the others do not. In this case, Sem still needs $O(K)$ packet transmissions in the forward direction. In the unlikely case that all nodes on the EGC have an event to report, Sem can lead to $O(K^2)$ packet transmissions in the forward direction as the straw-man protocol does in the backward direction. However, note that Sem requires only the node who sensed an event to send a report, and it is unusual that all nodes on the EGC send an event report in the same round. Therefore, the total number of packet transmissions in each round is usually much smaller with Sem than with the straw-man protocol, unless an attacked is detected.

Table 1 summarizes the packet transmission overhead in various cases. We can see that overhead incurred by Sem is similar to (or possibly higher than) that of the straw-man protocol in the round when an attack is detected. However, this cost is likely small compared to the overall system operation cost because the number of rounds when an attack is detected should be much smaller than the total number of rounds. Specifically, whenever an attack is detected, our protocol can identify the compromised node, which can then be removed, returning the system back to the normal mode.

In this analysis we have focused on packet transmission overhead and the gains due to Sem over the straw-man protocol. Note that there is also a corresponding gain in terms of computational overhead. In the forward direction, each node needs to verify only the signature of $N(0)$ on the PT in both the straw-man protocol and Sem (see in Section 5.2.4 that Sem does not require a normal node to verify the signature on the event report). In Sem, only the node to report an event signs on the PT. When sending an ACK in the backward direction, both protocols require a normal node to sign on the ACK. Verifying the signatures on the ACK is done by the BS. Again, Sem does not need this ACK when there is no attack. Here, recall that each signing and verification action with asymmetric cryptography is quite expensive – 2 s and 2.4 s with TinyECC in Micaz [6]. In Sem, most of the computation overhead is only incurred when there is an attack or when there is an event to report. Hence, the computational overhead is also significantly reduced.

7. Miscellaneous issues

7.1. Wormhole attacks

If two colluding compromised nodes $N(i)$ and $N(j)$ for any i, $j \in I\backslash\{0\}$, $j > i+1$, can talk to each other directly, $N(i)$ can send the PT to $N(j)$. If this is the case, the BS cannot get an event report from a node, say $N(k)$, between $N(i)$ and $N(j)$. To prevent this, the BS may randomly select a node each round, and make it send a null event report by notifying the command in the PT. When $N(k)$ is selected to do so, $N(i)$ has to send the PT to $N(i+1)$, but then $N(i+1)$ can find the round sequence Q, which is increased by more than 1 from the one that it has seen. This can be an indication that $N(i)$ is a compromised node. If $N(i)$ still directly forwards the PT to $N(j)$ in this case, $N(j+1)$ will find that the PT does not contain the null event report from $N(k)$, which can also

Table 1
Packet transmission overhead (F: forward direction; B: backward direction; K: the number of nodes on the event collection route).

Case		SEM		Straw-man protocol	
		F	B	F	B
No attack detected	No node to report an event	$O(K)$	0	$O(K)$	$O(K^2)$
	All nodes to report an event	$O(K^2)$	0	$O(K)$	$O(K^2)$
An attack detected	One node to report an event	$O(K)$	$O(K^2)$	$O(K)$	$O(K^2)$
	All nodes to report an event	$O(K^2)$	$O(K^2)$	$O(K)$	$O(K^2)$

be an indication that $N(j)$ is compromised. By making any node who first finds such an indication generate an error event report, we can provide the same security guarantee as in Proposition 4.

7.2. Jamming attacks

Note that a jamming attack may cause a node to stop circulating the PT or the ACK. If $N(i+1)$ is a victim of the jamming attack, i.e., it cannot successfully receive a PT or an ACK because a jamming signal keeps coming in, $N(i)$ will generate its own ACK. In this case, the BS thinks that $SET(i,i+1)$ is a suspicious set, which is wrong. Therefore, when we also consider the possibility of a jamming attack, we replace $SET(i,i+1)$ in the Proposition 1 with the set of the nodes within the jamming distance from the nodes $N(i)$ and $N(i+1)$. Then, it is easy to see that the statement of the proposition is still valid.

7.3. Persistent failures

By choosing a sufficiently large number of retransmissions in the ARQ mechanism, the packet loss probability due to transient node failures or link failures can be very small. However, it may not be zero in practice. Thus, we can still lose the PT or the ACK due to such transient failures, although chances are rare. This implies that in practice, $SET(i,i+1)$ identified by Proposition 1 may not include any compromised node. However, in such a case, $SET(i,i+1)$ can be regarded as a set of nodes that requires attention to repair. Thus, it is still useful to identify such a set from the network management point of view.

7.4. Formation of the EGCs

There would be many ways to form EGCs. The following is a simple example.

(Step 1) Let S denote the set of nodes in a network. The BS node creates a topology map of S, for example, using a link state routing protocol. Initially, all nodes are not marked.

(Step 2) Suppose that X is the farthest node from the BS among the unmarked nodes in S. Using Dynamic Source Routing (DSR) protocol, the BS node requests a route to X. The response from X may include the round-trip path between X and the BS. This round-trip path forms an EGC.[5] Mark the nodes in the EGC.

(Step 3) Repeat Step 2 until all nodes in S are marked.

As is commonly assumed in the literature, we assume that there is no compromised node at the beginning, and thus the formation of EGCs can be done with no attack at the beginning. In the Step 2, a node is allowed to mark more than once, which means that the node belongs to multiple EGCs.

8. Experiments

We have implemented SEM using TOSSIM [20], a popular sensor network simulator based on TinyOS [21]. For our experiments, we form three EGCs: EGC1 is comprised of 6 nodes ($K=6$), EGC2 8 nodes ($K=8$), and EGC3 10 nodes ($K=10$), as shown in Fig. 8. Note that SEM's security guarantees, e.g., Propositions 2 and 3 and Corollary 2, are not dependent on the locations of nodes, but only on the values of K and B. Thus, the locations chosen in Fig. 8 are only intended to show an example of how many retransmissions we need to recover a link failure for a given deployment.

The link gain between any two nodes is determined by a Java tool included in TinyOS v2.1, called LinkLayerModel [22], which models path loss and log-normal shadowing. We use a path loss exponent of factor 2.5, and the standard deviation of log-normal shadowing is 3 dB. By this model, the average receiving power at one-hop distance is sufficiently high for correct reception. For digital signature, we assume to use TinyECC [6] that takes about 2 s to gen-

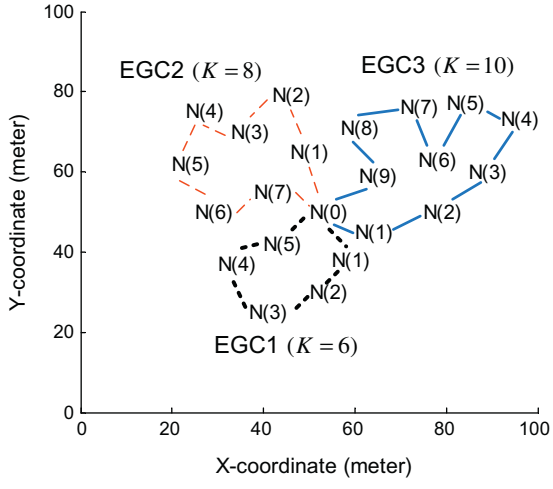


Fig. 8. Node locations in three EGCs for experiments: including the BS (i.e., $N(0)$), EGC1, EGC2, and EGC3 consist of 6, 8, and 10 nodes, respectively.

[5] This round trip path can contain the same node more than once in it, and so does the EGC. If a compromised node is included in the same EGC more than once as $N(i)$ and $N(j)$ for example, $N(i)$ and $N(j)$ should be considerred to be in collabortation.

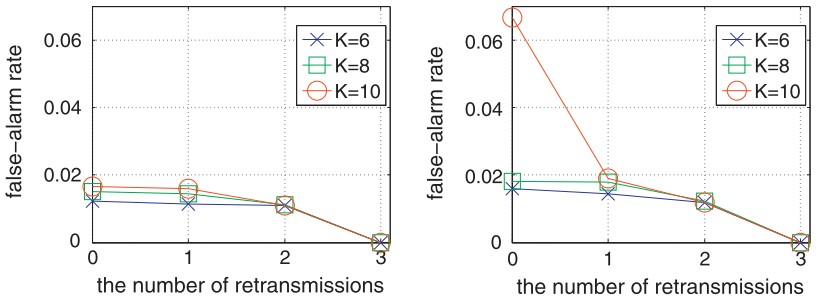


Fig. 9. False-alarm rate according to the number of retransmissions for the ARQ mechanism.

erate a signature and about 2.4 s to verify a signature. This TinyECC operation time mimics Micaz platform. Including the signature overhead and retransmission time (if necessary), the nodes hold the PT or the ACK less than 10 s, i.e., $B = 10$ s. Since Propositions 2 and 3 determine $T_{th} = -B_a(0)$ and $T_m = 2T_{th}$, we choose (T_{th}, T_m) in the unit of seconds as (110,220) for EGC1, (150,300) for EGC2, and (190,380) for EGC3. Thus, for example, in the EGC3, we regard $t_c < 190$ s as a legitimate case.

Fig. 9 shows the false-alarm rate as we vary the number of retransmissions. Here, the false-alarm rate is defined as the inverse of the number of successive rounds until we lose the PT due to natural causes, for example, due to a bad communication channel. Since in our experiments, we limit the number of rounds that we execute the simulation for to 100, if we do not lose the PT within 100 rounds, we report the false-alarm rate to be zero. The result in Fig. 9 shows that when we do not use the ARQ mechanism (i.e., no retransmission), the PT can be easily lost due to link failures. Hence, the false-alarm rate is high. In addition, the false-alarm rate is a little higher when we run three EGCs simultaneously than when there exists only one of the three EGCs. As we can expect, this is because the interference increases if the three EGCs work at the same time. However, when we allow the nodes to retransmit the PT up to three times on transmission failures, nearly all false-alarms are eliminated. Therefore, we conclude that at most three retransmissions would be a reasonable choice for the ARQ mechanism in our configuration.

In Fig. 10, we show the detection rate as a function of the amount of the delay introduced by a single compromised node. For the experiment, we select different positions for the compromised node: $N(2)$ and $N(4)$ for EGC1, $N(2)$, $N(4)$, and $N(6)$ for EGC2, and $N(2)$, $N(5)$, and $N(8)$ for EGC3. As we mentioned earlier, the compromised nodes may hold the PT longer than $B = 10$ s without being detected, since the BS will not complain unless the circulation time t_c is larger than the threshold T_{th}. Fig. 10a–c shows such a situation. For example, if a compromised node in EGC3 holds the PT for 110 s, the BS still receives the PT back within T_{th}, and thus the compromised node never gets detected. However, when the delay added by the compromised node is between 110 and 170 s, the circulation time t_c may or may not be larger than the threshold T_{th}, depending on how long the other nodes hold the PT. We can see from Fig. 10c that if the amount of the malicious delay is over 170 s, it is always detected. Here, remember that once the malicious activity is detected, the BS will not cancel the ACK timers at the sensor nodes, thus gathering an ACK from the nodes. Although we do not show in this figure, once detected, the compromised node is always identified in a form of a suspicious set, as we proved. Further, the amount of malicious delay that the compromised node can introduce without being detected is always the amount of time that is left over in T_{th} after the nodes actually spent their own time to transmit the PT. Thus, if we reduce the value of B, and thus reduce the value of T_{th}, then the compromised node must also decrease malicious delay to avoid being detected. We can see this effect from Fig. 10d and 10e, which is obtained from almost the same configuration as Fig. 10a–c, except that the value of B is changed from 10 s to 1 s.[6] We can also see from the figures that the position of a single compromised node does not affect the detection rate as is to be expected since the detection is only dependent on the cumulative time for circulating the PT in a EGC.

In Fig. 11, we study the case when there exist multiple compromised nodes in EGC2 that delay the PT. We are interested in finding out how the positions of these compromised nodes will determine which one of them is identified by SEM. Recall from Proposition 1 that if a node $N(i)$ generates an ACK, it will be included in the suspicious set. In Fig. 11, the y-axis normalized frequency represents the number of the ACKs generated by $N(i)$, divided by the total number of cases that BS finds $t_c > T_{th}$. For this experiment, we assume that $N(2)$, $N(4)$, and $N(6)$ are malicious in EGC2, and each of the three nodes holds the PT for $x/3$ s in order to cause x seconds of accumulated delay. We set these delays to be identical so that we can focus on how the position of the compromised node affects the probabil-

[6] It is possible to set $B = 1$ s if we assume the use of the Imote2 platform that can perform a signature operation within a few tens of milliseconds [6].

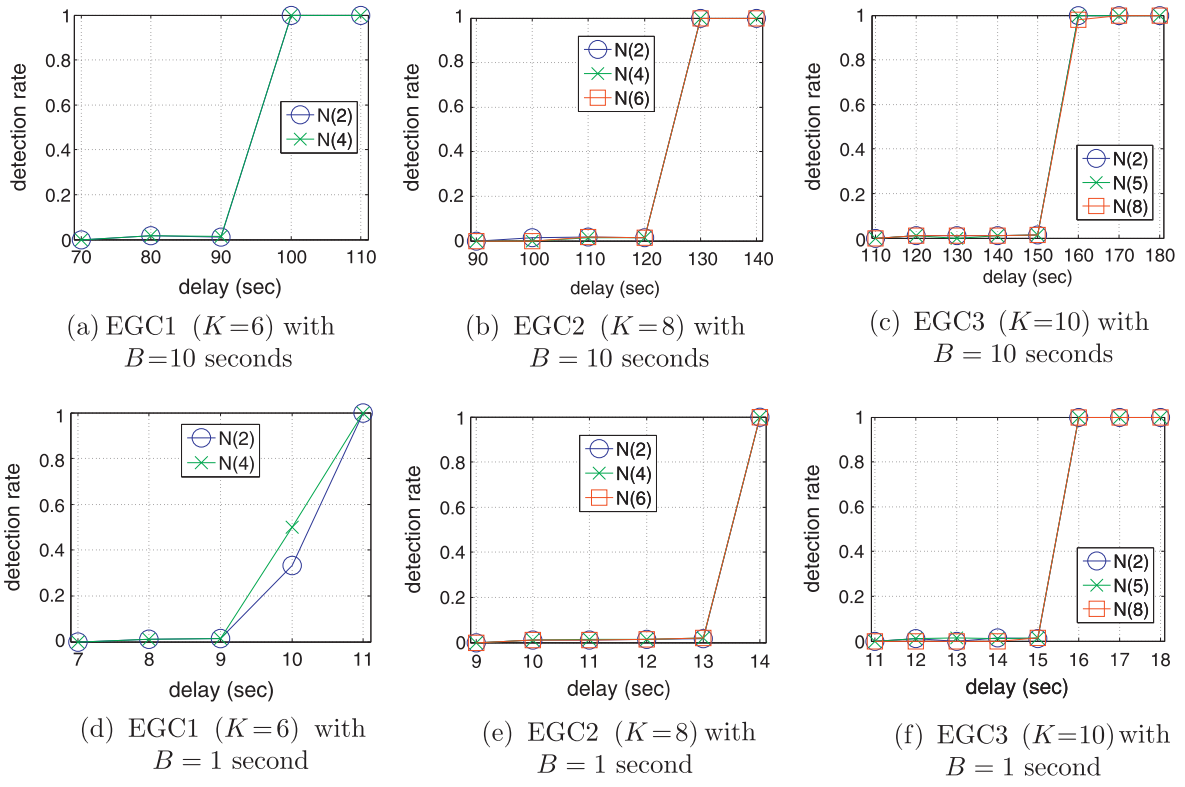


Fig. 10. Detection rate according to the delay introduced by a single compromised node.

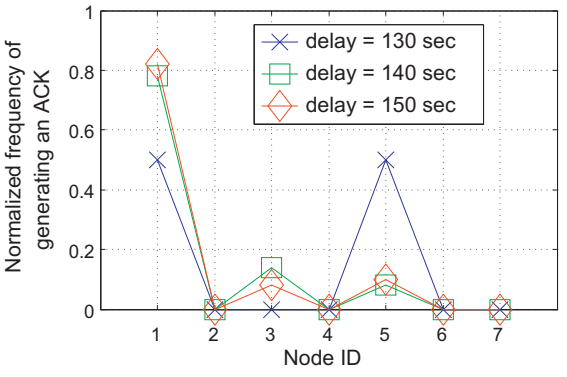


Fig. 11. Normalized frequency of generating an ACK in EGC2 ($K = 8$) when there are multiple compromised nodes that delays the PT. Here, we assume that $N(2)$, $N(4)$, and $N(6)$ are compromised nodes. For x seconds of given delay in legend, each compromised node delays the PT for $x/3$ s.

ity of being identified, when several compromised nodes delay the PT in collusion. Note that as shown in Fig. 10b, when the accumulated delay x is equal to or larger than 130 s in EGC2, we will be able to identify one of the compromised nodes. Further, in our simulation setting, the malicious nodes are assumed to generate an ACK only if necessary, i.e., only when their ACK timer expires. That is, the malicious nodes wait for the ACK from other nodes until their ACK timer expires. We do not allow them to generate an ACK anytime before the expiration of a ACK timer, because in that case they directly reveal that they are malicious (by being included in a suspicious set). We can make a number of observations from Fig. 11. First, note that BS may receive the ACK that is generated by $N(1)$, $N(3)$, or $N(5)$. In all the cases, one of malicious nodes $N(2)$, $N(4)$, and $N(6)$ is included in a suspicious set. Second, we can see that although one among the three malicious nodes is always located, the node being identified may vary depending on the accumulated delay. The larger the accumulated delay is, the more likely that the node closer to BS is identified. This is because larger delay leaves less time in the ACK timer till expiration at each node, and the node closer to BS needs to wait longer to receive the ACK generated by $N(K)$.

9. Conclusion

In event monitoring, a number of sensor nodes are deployed over a region where some phenomenon is to be monitored. When an event is detected, the sensor nodes report it to a base station, where a network operator can take appropriate action using the event report. However, such an event reporting process can be easily attacked by compromised nodes in the middle that drop, modify, or delay the report packet. No prior work has been able to provide a security guarantee of timely and reliable collection of event reports at a base station *in the presence of Byzantine adversarial nodes that are capable of colluding among themselves.*

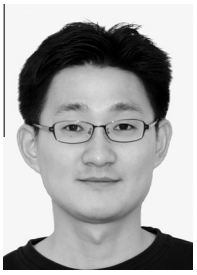
To resolve this issue, we have presented SEM, a secure event monitoring protocol in the face of Byzantine adversaries. SEM provides a strong and useful guarantee that, whenever the compromised nodes launch an attack and causes the event report from a legitimate node not to reach the BS within a bounded time, the BS can identify a pair of nodes that are guaranteed to contain at least one compromised node. As a result, the system can defeat the attack in the sense that the network operator can remove or reprogram the detected compromised node one by one, eventually securing a safe route to the BS for collecting an event report. To the best of our knowledge, we are not aware of any other protocols in the literature that can provide similar security guarantees. Hence, we believe that our gain in security is significant.

SEM is designed to reduce the overhead due to asymmetric cryptographic operations in terms of computational overhead and additional network packets that are generated: it uses low overhead in normal scenarios when there is no attack, and only invokes the heavier overhead when an attack is launched.

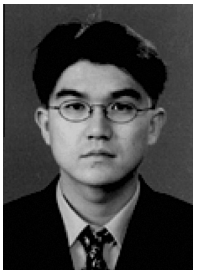
Our future work will focus on having SEM work under more dynamic scenarios, e.g., where nodes may move or be incrementally introduced into the network, and making SEM even more resource efficient.

References

[1] J. Carle, D. Simplot-Ryl, Energy-efficient area monitoring for sensor networks, Computer 37 (2004) 40–46.
[2] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, D. Culler, Design of a wireless sensor network platform for detecting rare, random, and ephemeral events, in: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN '05, IEEE Press, 2005.
[3] A. Giani, E. Bitar, M. Garcia, M. McQueen, P. Khargonekar, K. Poolla, Smart grid data integrity attacks: characterizations and countermeasuresπ, in: 2011 IEEE International Conference on Smart Grid Communications (SmartGridComm), IEEE, 2011.
[4] Perimeter Security Alarm System. <http://www.hkvstar.com/>.
[5] R.K. Panta, S. Bagchi, S.P. Midkiff, Efficient incremental code update for sensor networks, in: ACM Trans. Sen. Netw, ACM, 2011.
[6] A. Liu, P. Ning, TinyECC: a configurable library for elliptic curve cryptography in wireless sensor networks, in: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN'08, 2008.
[7] T. He, S. Krishnamurthy, J.A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, Energy-efficient surveillance system using wireless sensor networks, in: Mobisys, ACM Press, 2004, pp. 270–283.
[8] T. Yan, T. He, J.A. Stankovic, Differentiated surveillance for sensor networks, in: Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems, SenSys'03, 2003.
[9] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: Proceedings of the 33rd Hawaii International Conference on System Sciences, HICSS '00, IEEE Computer Society, Washington, DC, USA, 2000.
[10] P. Rothenpieler, D. Krger, D. Pfisterer, S. Fischer, D. Dudek, C. Haas, M. Zitterbart, FleGSens – secure area monitoring using wireless sensor networks, in: World Academy of Science, Engineering and Technology, 2009.
[11] N.G. Duffield, M. Grossglauser, Trajectory Sampling for Direct Traffic Observation (2001).
[12] J. Sommers, P. Barford, N. Duffield, A. Ron, Improving accuracy in end-to-end packet loss measurement, in: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '05, 2005.
[13] J. Sommers, N. Duffield, Accurate and efficient SLA compliance monitoring, in: Proceedings of ACM SIGCOMM, 2007.
[14] S. Goldberg, D. Xiao, E. Tromer, B. Barak, J. Rexford, Path-quality monitoring in the presence of adversaries, in: ACM SIGMETRICS, 2008.
[15] S. Marti, T. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: MOBICOM, 2000.
[16] I. Khalil, S. Bagchi, N.B. Shroff, Slam: Sleep-Wake Aware Local Monitoring for Sensor Networks, 2006.
[17] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, H. Rubens, ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks, ACM Transactions on Information and System Security 10 (4) (2008) 1–35.
[18] X. Zhang, A. Jain, A. Perrig, Packet-dropping adversary identification for data plane security, in: Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT '08, 2008.
[19] A.A. Pirzada, C. McDonald, Kerberos assisted authentication in mobile ad-hoc networks, in: Proceedings of the 27th Australasian Conference on Computer science, vol. 26, ACSC '04, 2004.
[20] TOSSIM Tutorial. <http://docs.tinyos.net/index.php/TOSSIM>.
[21] TinyOS Documentation Wiki. <http://docs.tinyos.net/index.php>.
[22] Building a Network Topology for TOSSIM. <http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/usc-topologies.html>.



Jinkyu Koo received his B.E. degree in Electrical Engineering from Korea University, Seoul in August 2001, his M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST) in February 2004, and his Ph.D. degree from Purdue University, West Lafayette, Indiana in August 2012. His research interests include embedded systems, sensor networks, network security, optimization, and operating systems.



Dong-Hoon Shin is currently a Ph.D. candidate in the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana. He received his B.E. and M.S. degrees in Electrical Engineering from Korea University in 2003 and from Korea Advanced Institute of Science and Technology (KAIST) in 2006, respectively. He has worked as an intern at Intel in Hillsboro, Oregon, in Standards and Advanced Technology Group, from August 2011 to June 2012. His research interests lie in the areas of wireless networks, mobile systems and smart grids with emphasis on mathematical modeling, algorithm/protocol design and performance analysis for security of these systems.

Xiaojun Lin received his B.S. from Zhongshan University, Guangzhou, China, in 1994, and his M.S. and Ph.D. degrees from Purdue University, West Lafayette, Indiana, in 2000 and 2005, respectively. He is currently an Associate Professor of Electrical and Computer Engineering at Purdue University.

His research interests are in the analysis, control and optimization of wireless and wireline communication networks. He received the IEEE INFOCOM 2008 best paper award and 2005 best paper of the year award from Journal of Communications and Networks. His paper was also one of two runner-up papers for the best-paper award at IEEE INFOCOM 2005. He received the NSF CAREER award in 2007. He was the Workshop co-chair for IEEE GLOBECOM 2007, the Panel co-chair for WICON 2008, the TPC co-chair for ACM MobiHoc 2009, and the Mini-Conference co-chair for IEEE INFOCOM 2012. He is currently serving as an Area Editor for (Elsevier) Computer Networks journal, and has served as a Guest Editor for (Elsevier) Ad Hoc Networks journal.

Saurabh Bagchi is a Professor in the School of Electrical and Computer Engineering and in the Department of Computer Science (by courtesy) at Purdue University, West Lafayette, Indiana. He is a senior member of IEEE and ACM, and an ACM Distinguished Speaker. At Purdue, he is the Assistant Director of CERIAS, the security center, and leads the Dependable Computing Systems Laboratory (DCSL), where his group performs research in practical system design and implementation of dependable distributed systems. He is an IMPACT Faculty Fellow at Purdue and a Visiting Scientist at IBM Austin Research Lab.