Secure Time Synchronization Service for Sensor Networks

Saurabh Ganeriwal, Srdjan Capkun, Chih-Chieh Han, Mani B. Srivastava
Networked and Embedded Systems lab, 56-125B, EE-IV, University of California Los Angeles
{saurabh, simonhan}@ee.ucla.edu, {capkun, mbs}@ucla.edu

ABSTRACT
In this paper, we analyze attacks on existing time synchronization protocols for wireless sensor networks. We propose a secure time synchronization toolbox to counter these attacks. This toolbox includes protocols for secure pairwise and group synchronization of nodes that lie in each other's power ranges and of nodes that are separated by multiple hops. We provide an in-depth analysis of security and energy overhead of the proposed protocols.

Categories and Subject Descriptors
C.2.2 [**Computer Systems Organization**]: Computer Communication Networks – *Network Protocols.*

General Terms
Algorithms, Security, Verification.

Keywords
Sensor Networks, Security, Time Synchronization, Message Authentication Code, Delay.

1. INTRODUCTION
Time synchronization is critical to sensor networks at many layers of its design. It enables better duty-cycling of the radio, accurate and secure localization, beamforming and other collaborative signal processing. Examples of existing sensor network applications where precise time is needed include: measuring the time-of-flight of sound; distributing an acoustic beam forming array; forming a low-power TDMA radio schedule; integrating a time-series of proximity detections into a velocity estimate; suppressing redundant messages by recognizing duplicate detections of the same event by different sensors; ordered logging of events during system debugging; integrating multi sensor data; or coordinating on future action. Imagine the detrimental affect on the functionality of all these applications if a malicious adversary is able to abuse the underlying time synchronization protocol. Nodes will have faulty estimates about the location of other nodes. Packets will be lost if the sleep-wakeup schedules of nodes do not intersect. This can further trigger unnecessary packet retransmissions if MAC layer acknowledgements are enabled. It will be trivial for adversaries to perform replay attacks. Collaborative data processing and signal processing techniques will be adversely affected.

WiSE'05, September 2, 2005, Cologne, Germany.

Although time synchronization problem has been thoroughly studied in sensor networks [1] and there are several prototype implementations, such as RBS [2], TPSN [3], FTSP [4] that can achieve microseconds accuracy, none of the existing protocols are resilient to malicious attacks. These protocols have not been built with security in mind. Realizing the inadequacy of existing time synchronization solutions, we develop several schemes for achieving secure time synchronization in sensor networks. Our approach involves integrating security mechanisms into existing protocols as well as developing new protocols from scratch. Our contributions are three fold.

First, we perform an in-depth security analysis of the sender-receiver synchronization protocols [2], [3]. We show that as sensor networks are deeply coupled with the physical world that they monitor, a malicious adversary can subvert the time synchronization protocol by exploiting weaknesses at the interface between the sensor network and the physical world. Examples of time synchronization protocols vulnerable to these attacks include TPSN [3], FTSP [4], LTS [5], Mini/Tiny Sync [6]. Although exactly the same attacks are feasible on receiver-receiver synchronization [3], we do not explicitly cover them in this paper due to space constraints.

Second, we integrate security mechanisms into the basic approach of sender-receiver synchronization to propose a protocol for secure pairwise time synchronization in sensor networks. We show that for a nominal overhead, our protocol can counter malicious attacks from external attackers. We extend this basic scheme to propose and analyze three protocols for secure pairwise synchronization over multiple hops: opportunistic, direct and transitive. Each of these protocols offers a different point of operation in the energy – security subspace and should be used according to the application needs.

Third, we propose a protocol for secure group time synchronization. We show that this protocol is resilient to attacks from external attacker as well as to attacks from a subset of compromised group nodes. Typical applications of secure group synchronization include object tracking, beamforming, intruder detection and fire monitoring. These applications will function accurately only if the synchronization error between any two nodes in a monitoring group is bounded within some pre-specified limits. In our protocol this is achieved, in part, through *data fusion*, the process of transforming and merging individual sensor readings into a high-level sensing result.

In this paper, we focus on detecting the malicious attacks. We abort the ongoing time synchronization protocol once the attack is detected. We neither provide mechanisms to prevent the attacks from taking place nor do we provide mechanisms to continue with the process of time synchronization while modifying the protocol taking into account the malicious attack. Both of these are challenging problems and form part of our future work. Yet, a significant contribution of this paper is to highlight the security

flaw of existing time synchronization protocols and prevent the sensor nodes from incorrectly assuming that they are synchronized when in reality they are not.

2. PROBLEM FORMULATION

Our system consists of a network of sensor nodes. If two sensor nodes reside within the power range of one another, they are considered neighbors. We assume that the radio link between neighbors is bidirectional. We also assume that neighboring sensors share pairwise secret keys. There are several schemes in literature for secure pairwise key establishment between sensor nodes [7], [8], [9].

2.1 Sensor Node Clock

Every sensor node maintains its own clock and this is the only notion of time that a node has. The clock is an ensemble of hardware and software components; it is essentially a timer that counts the oscillations of a quartz crystal running at a particular frequency. Let us represent this clock for node A by C_A. Furthermore $C_A(t)$ represents the time in the local clock of node A at real time t. This real time can be UTC. In this paper, we are not concerned about synchronizing the clocks with real time; we will use it as a common reference to compare different sensor node clocks. The difference in the clocks of two sensor nodes at any time t is referred as the *offset* error between them.

There are three reasons for the nodes to be representing different times in their respective clocks – (1) The nodes might have been started at different times, (2) The quartz crystals at each of these nodes might be running at slightly different frequencies, causing the clock values to gradually diverge from each other (termed as the *skew* error), (3) The frequency of the clocks can change variably over time because of aging or ambient conditions such as temperature (termed as the *drift* error). Figure 1 summarizes these sources of error using a representative example of two nodes, A and B.

1. Offset (δ) = $C_A(t) - C_B(t)$

2. Skew (η) = $\frac{\partial C_A}{\partial t} - \frac{\partial C_B}{\partial t}$

3. Drift (λ) = $\frac{\partial^2 C_A}{\partial t^2} - \frac{\partial^2 C_B}{\partial t^2}$

Figure 1. Sources of Error

In this paper, we aim to achieve instantaneous synchronization between sensor nodes. So our aim is to remove the offset error between the nodes at any given instant of time.

2.2 Sender-Receiver Synchronization

The approaches for synchronizing a pair of nodes can be broadly classified as sender-receiver or receiver-receiver [3]. Pairwise sender-receiver synchronization is performed by a handshake protocol between a pair of nodes. This protocol is executed in three steps as follows[1]:

Pairwise Sender-receiver Synchronization
1. A(T1) → (T2)B : A, B, sync
2. B(T3) → (T4)A : B, A, T2, T3, ack
3. A calculates offset between the nodes.

[1] We use the following notation throughout this paper.

Node-id (Send time) → (Receive time) Node-id : Packet contents.

Here, $T1$, $T4$ represent the time measured by the local clock of node A, C_A. Similarly $T2$, $T3$ represent the time measured by C_B. At time $T1$, A sends a synchronization pulse packet to B. Node B receives this packet at $T2$, where $T2$ is equal to $T1+\delta+d$. Here, δ and d represent the offset between the two nodes and end-to-end delay respectively. At time $T3$, B sends back an acknowledgement packet. This packet contains the values of $T2$ and $T3$. Node A receives the packet at $T4$. Similarly, $T4$ is related to $T3$ as $T4 = T3-\delta+d$. Node A can now calculate the clock offset and the end-to-end delay as:

$$\delta = \frac{(T2-T1)-(T4-T3)}{2}; d = \frac{(T2-T1)+(T4-T3)}{2} \ldots (1)$$

Several schemes such as TPSN [2], LTS [5], Mini/Tiny Sync [6] are based on sender-receiver synchronization.

2.3 Malicious Attacks

In this section, we list some of the attacks that an external malicious attacker can perform to abuse the functionality of the pairwise sender-receiver synchronization protocol. An external attacker will be successful if he makes node A calculate a faulty value of the offset.

- The attacker can modify the values of $T2$ and $T3$. It can change the contents of the message being sent from B to A or it can generate a completely new message by assuming the identity of node B.
- Besides directly modifying the values of $T2$ and $T3$, the attacker can influence their measurement. Notably, $T2$ is measured as the time at which the initial synchronization pulse packet sent by A is received at B. If an attacker could delay the time at which B receives the synchronization pulse, he could modify the computation of the offset at A. To delay the synchronization pulse, an attacker can simply jam the initial pulse, store in its memory and replay it at some arbitrary time in the future as shown in Figure 3. This attack cannot be prevented by the use of conventional cryptographic primitives. From now onwards, we refer to this attack as the *pulse-delay* attack. Here, we assume that the attacker can jam the communication between two nodes by transmitting signals which will disrupt packet reception at the receiver. By jamming, we consider disruptive jamming that cannot be detected at the receiver. Currently available sensor network platforms use 433MHz Chipcon1000, 2.4 GHz IEEE 802.15.4 compliant (Direct Sequence (DSSS)) or Bluetooth (Frequency Hopping (FHSS)). Even if DSSS and FHSS resist various types of jamming well, because of their low transmitting RF power (1mW), these sensor platforms are vulnerable to broadband jamming (BBN). Recently, Xu et al. [10] showed that jamming attacks are indeed feasible against Mica2 motes, and that detecting these attacks requires significant resources.
- The attacker can perform a similar pulse-delay attack on the acknowledgement packet to modify $T4$.

2.3.1 Pulse-Delay Attack

Figure 2 shows the idea behind pulse-delay attack. We have already shown that we use two equations to derive the end-to-end delay and the clock offset between the sender and the receiver. These are: $T2= T1+\delta+d$ and $T4 = T3-\delta+d$. If an attacker performs

a pulse-delay attack (e.g., on the initial sync packet), the equations will change to: $T2^* = T1+\delta+d$ and $T4 = T3-\delta+d$, where $T2^* = T2+\Delta$. Here Δ is the pulse-delay introduced by the attacker.

The clock offset and the end-to-end delay then become:

$$\delta = \frac{(T2-T1)-(T4-T3)+\Delta}{2}; d = \frac{(T2-T1)+(T4-T3)+\Delta}{2} \ldots (2)$$

This shows that the attacker can, by varying the pulse-delay, Δ, arbitrarily change the computed clock offset. An important observation is that in the process of carrying out a pulse-delay attack, the attacker also changes the computed end-to-end delay. In equation (2) both offset, δ, and the end-to-end delay, d, are higher by $\Delta/2$ than equation (1). Note that the attacker has no control over the calculation of the end-to-end delay. As we will show in later sections, we exploit this observation to develop a protocol for detecting pulse-delay attacks.

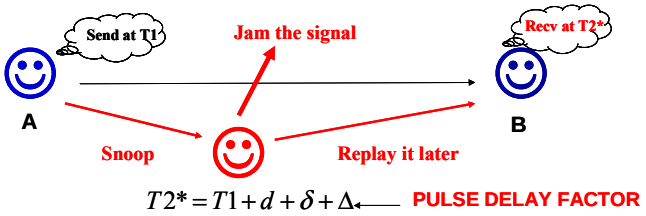

Figure 2. Pulse-delay attack

3. SECURE TIME SYNCHRONIZATION

In this section, we integrate security mechanisms into the basic approach of sender-receiver synchronization to make it resilient to adversarial attacks from external attackers.

3.1 Secure Pairwise Synchronization (SPS)

We propose the following protocol:

Secure Pairwise Synchronization (SPS)
1. A (T1) → (T2) B : A, B, N_A, sync
2. B (T3) → (T4) A :
   B,A,N_A,T2,T3,ack, MAC{K_{AB}}[B,A,N_A,T2,T3,ack]
3. A calculates end-to-end delay d={(T2-T1)+(T4-T3)}/2
   If d≤d* *then* δ={(T2-T1)-(T4-T3)}/2, *else* abort

In this protocol, message integrity and authenticity are ensured through the use of Message Authentication Codes (MAC) and a key K_{AB} shared between A and B. This prevents external attackers from successfully modifying any values in the synchronization pulse or in the acknowledgement packet. Furthermore, the attacker cannot assume an identity of node B as it does not hold the secret key K_{AB}. An attacker can hear the packet over the wireless channel and can use the MAC in the future to generate authenticated packets. Using a random nonce, N_A, during the handshake safeguards the protocol against such replay attacks.

Potentially more harmful attacks are pulse-delay attacks. In our protocol, these attacks are detected through a comparison of the computed message end-to-end delay, d, with the maximal expected message delay d^*. Note that the calculation of the end-to-end delay, d, comes as an auxiliary benefit of the protocol. We have added no extra overhead on the functionality of sender-receiver synchronization. If the computed delay is greater than the maximal expected delay, we abort the offset calculation.

3.2 Performance Evaluation

Clearly, how much the attacker can influence the synchronization relies on our estimate of the maximal delay d^* and in order to make a judicious choice let us first analyze the end-to-end delay, d, in the absence of any external attackers. The three significant contributors to the end-to-end delay are:

1. Waiting time at the medium access control (mac[2]) layer to access the channel: This delay is variable in nature and can range from a few microseconds to a few minutes.
2. Time taken in transmitting the packet bit-by-bit at the radio of the sender node: This time will be in hundreds of microseconds and is deterministic in nature. It depends on the packet size and the radio speed.
3. Propagation time over the wireless link between the sender and receiver node: This is only a few nanoseconds.

Time stamping the packets below the mac layer is feasible on typical sensor networking platforms [3], which removes the most significant variable factor, mac access waiting time. The only other variable component is the propagation time over the wireless link. The transmission delay only depends on the size of the packet which can be fixed. Note that although the relevant contents in the sync and ack packets are of different lengths, we add some redundant bits to each packet to ensure that they are of the same length. This ensures that the transmission delay is symmetric for the two packet exchanges. Since the transmission delay is many orders of magnitude larger than the end-to-end delay, a stable value of the end-to-end delay can be calculated.

3.2.1 Measurement on Mica2 Motes

Timing-sync protocol for sensor networks (TPSN), proposed in [3], is one of the most popular approaches for time synchronization in sensor networks that is based on sender-receiver synchronization. This protocol uses mac layer time stamping to achieve an accuracy of around 10μs for mica2 motes. We needed to incorporate the cryptographic functionality in this protocol. Specifically, as the packets are time stamped below the mac layer, we needed a cryptographic library that can calculate the Message Authentication Code (MAC) on-the-fly as the packets are being transmitted. TinySec [19], a symmetric cryptographic library on motes, enables this MAC calculation. Essentially, our prototype implementation of SPS integrates the time stamping library provided by TPSN with TinySec and performs a thresholding on the computed delay at the end of the two-way packet exchange.

We ran this prototype implementation of SPS on 5 different pair of motes. For every pair we ran multiple independent runs to calculate the value of the computed delay (d in equation (1)) from the protocol. In all, we computed the delay for 1000 independent runs. The first plot in Figure 3 shows the actual delay measured in every run and the second plot shows the corresponding histogram of the measured delay. Table 1 summarizes the statistics.

[2] We use lowercase for Medium Access Control (mac) and uppercase for Message Authentication Codes (MAC).

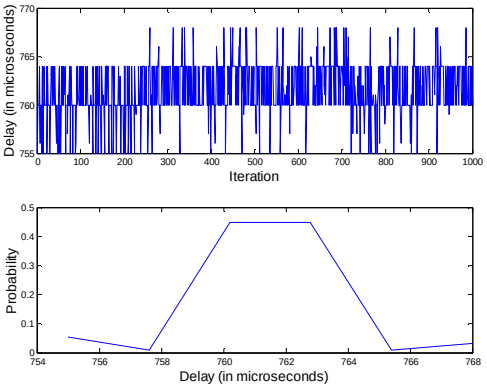

Figure 3. End-to-end delay over a single link

Table 1: Statistics of end-to-end delay over a link

Maximum (µs)	Minimum (µs)	Average (µs) (d_{avg})	Standard deviation (σ)
768	755	762	2.82

The histogram of the computed delay closely resembles a Gaussian distribution. This is consistent with the results reported by authors in [2], [3]. Since the end-to-end delay follows a Gaussian distribution, $d \approx N(d_{avg}, \sigma)$, with a 99.97% confidence the delay will fall in the interval $[d_{avg} - 3\sigma, d_{avg} + 3\sigma]$. Hence, the maximal delay should be set to $d_{avg} + 3\sigma$, i.e., $d^* = d_{avg} + 3\sigma = 762 + 3*2.82 \approx 771\mu s$.

3.2.2 Minimum Synchronization Precision

If the end-to-end delay was a constant, we should have been able to synchronize the nodes perfectly (zero error). The delay variation introduces synchronization error. Intuitively, the minimum synchronization precision (maximum error) will occur when the end-to-end delay variations in the two directions (from *A* to *B* and from *B* to *A*) are maximal, i.e., in one direction *d* is equal to $d_{avg} - 3\sigma$ and in the other direction *d* is equal to $d_{avg} + 3\sigma$. Using equations (1) and (2), the synchronization error for this worst-case scenario will be 3σ (around 10µs).

3.2.3 Maximum Attacker Impact

We define the maximum attacker impact to be the maximum difference between the clocks of two nodes that the attacker can cause without getting detected. The worst case will occur when the actual end-to-end delay is equal to the minimum, $d_{avg} - 3\sigma$. In this scenario, the attacker can introduce a maximum pulse-delay factor of 12σ ($\Delta=12\sigma$). Node *A* will calculate the end-to-end delay, *d*, as:

$$d = d_{actual} + \Delta/2 = d_{avg} - 3\sigma + (12\sigma/2) = d_{avg} + 3\sigma = d^* \ldots (3)$$

As the computed end-to-end delay is equal to the maximal delay, the thresholding verification will pass. Using equation (2), the calculated offset will be off by $\Delta/2$. Thus, the maximum attacker impact is 6σ (around 20µs).

3.2.4 Discussion

In the absence of any malicious behavior, SPS is able to achieve the same accuracy as TPSN. The extra computation overhead of MAC calculation does not impact the accuracy of time synchronization as we are able to do on-the-fly MAC calculation. This parallelism will not be feasible on emerging new 802.15.4 compliant platforms such as Micaz, Telos [20] etc., where the radio speeds are higher than Mica2 motes by an order of magnitude. However, 802.15.4 standards also mandate the implementation of the cryptographic library (equivalent of TinySec) in hardware. Specifically, AES is available in hardware providing a faster interface. Furthermore, 802.15.4 compliant radios expose a packet interface to the application layer unlike the byte interface exposed by the chipcon radios in Mica2 platforms. These different set of attributes opens up an interesting design challenge. The implementation of SPS for this new class of platforms forms the part of our future work.

4. MULTIHOP SYNCHRONIZATION

So far, we have assumed that the two nodes that need to synchronize are in each-other's power range and thus can directly communicate. In this section, we propose and analyze three protocols for secure sender-receiver synchronization over multiple hops: opportunistic, direct and transitive. We assume that two sensors can obtain sets of communication paths between them, either through a priori knowledge of network topology, through topology discovery [11], or through routing information [12].

We present the protocols through a representative example: *A – C – D – B*. In this example, node *A* wants to synchronize to node *B* which is not in its direct communication range. In fact the shortest path between these two nodes comprises of three hops, going through nodes *C* and *D*.

4.1 Secure Opportunistic Multi-hop (SOM)

The SOM protocol is executed as follows:

Secure Opportunistic Multi-hop Synchronization (SOM)
1. A (T1) → C → D → (T2) B : A, B, N_A, sync
2. B (T3) → D → C → (T4) A : B,A,N_A,T2,T3,ack, MAC{K_{AB}}[B,A,N_A,T2,T3,ack]
3. A calculates end-to-end delay d={(T2-T1)+(T4-T3)}/2
   If $d \le d_M^*$ *then* δ={(T2-T1)-(T4-T3)}/2, *else* abort

This protocol is essentially the same as the protocol used for one-hop synchronization, with a difference that in this protocol we assume that there are several forwarding nodes between the sender and the receiver. Note that this protocol assumes that there exist a secret key between *A* and *B*, K_{AB}, which are multiple hops apart.

The expected end-to-end delay *d*, computed at *A* will be much longer in this protocol than in the one-hop case. We take this into account by estimating a maximum expected (and allowed) end-to-end delay by a larger value $d_M^* >> d^*$. Like in the one-hop time synchronization protocol, the variance of d_M^* will determine how fine-grained synchronization is and will upper-bound attacker's possible impact on the synchronization precision.

The end-to-end delay in SOM is equal to the cumulative sum of the transmission delays between each pair of nodes on the path and the mac access delays of the forwarding nodes. Here, the processing delays at nodes can be neglected, as they are two to three orders of magnitude smaller than transmission and mac access delays. Although the transmission delays can be accurately predicted from radio speed, mac access times can be very unpredictable and can range from microseconds to a few minutes depending on the condition of the channel. Furthermore, channel

access times also depend on the intensity of the communication between the nodes in the network, which is application-specific.

4.1.1 Performance Evaluation

We performed an empirical study to find out the delay variations over multiple hops in a typical sensor network. The results were obtained in simulations using the Avrora simulator [13]. Avrora provides a cycle-accurate simulation of the AVR microcontroller, allowing real programs to be run with precise timing. We use the same TinyOS code – network stack, time stamping, etc. as in the previous section to obtain these results. Table 2 summarizes the results for varying number of hops between the sender and receiver. The statistics were obtained from 1000 independent runs.

Table 2: Statistics of the end-to-end delay over multiple hops

Hop distance	Maximum (µs)	Minimum (µs)	Average (µs) $(d_{avg})_M$	Standard deviation (µs) (σ_M)
2	32094	18761	25120	3861
3	62926	37510	49940	5450
4	92509	56260	74781	6738
5	120841	76259	99667	7827
6	149174	97092	124393	8841

It can be observed from Table 1 and 2 that the average delay and the standard deviation for the multi-hop case are significantly higher, by three orders of magnitude, than the corresponding numbers over a link (single hop). We observed that the end-to-end delay over multiple hops also follows a Gaussian distribution. Given this, the same analysis can be worked out for SOM as in SPS. The maximal delay should be set to $d_M^* = (d_{avg})_M + 3\sigma_M$. The minimum synchronization precision and the maximum attacker impact will be given by $3\sigma_M$ and $6\sigma_M$ respectively. Figure 4 plots these two quantities for varying number of hops. Note that the values are significantly higher, by three orders of magnitude, than the SPS protocol.

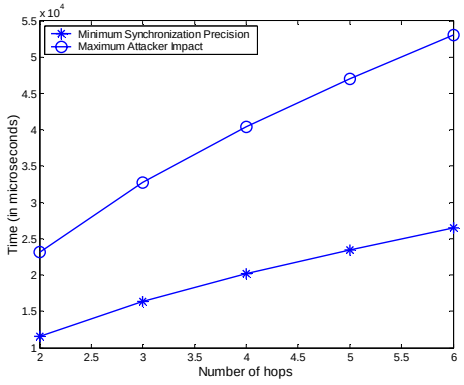

Figure 4. Secure Opportunistic Multi-hop Synchronization

We would like to point out that the values of $(d_{avg})_M$ and σ_M, as shown in Table 2, will change with changes in the network topology and the underlying communication traffic. The objective of this empirical study is to give representative numbers so that a meaningful relative comparison can be made with the other protocols. Even if SOM protocol provides very low synchronization accuracy, it can provide protection against external as well as internal attackers. This is because the intermediate nodes in SOM do not perform any processing on the packet. They simply receive the packet and forward it to the next hop node. Thereby, even if node *D* has been compromised and the attacker tries to modify the value of *T2*, *T3* at node *D*, it will result in a MAC verification failure at node *A*.

In the following sections, we propose two protocols – Secure Direct Multi-hop Synchronization (SDM) and Secure Transitive Multi-hop Synchronization (STM) that provide much better accuracy than SOM. However, both of them assume that intermediate nodes (*C* and *D* in the example) are trustworthy and hence, the protocols are not resilient to attacks from compromised nodes.

4.2 Secure Direct Multi-hop (SDM)

The SDM protocol is executed as follows:

Secure Direct Multi-hop Synchronisation (SDM)

1. A (T1) → (T2)C(T3) → (T4)D(T5) → (T6) B : A,B,N_A,sync
2. B : m1={B,A,T6,T7,ack}
   : M_1= MAC{K_{BD}}[B, D, N_A, m1]
   B (T7) → (T8) D: B, D, N_A, m1, M_1
3. D : m2={B,D,A,T4,T9,(T6-T5),(T8-T7),ack}
   : M_2= MAC{K_{DC}}[D, C, N_A, m2]
   D (T9) → (T10) C: D, C, N_A, m2, M_2
4. C : m3={B,D,C,A,T2,T11,(T4-T3),(T10-T9),(T6-T5),(T8-T7),ack}
   : M_3= MAC{K_{CA}}[C, A, N_A, m3]
   C (T11) → (T12) A: C, A, N_A, m3, M_3
5. A : calculate

$$d = \frac{\{(T2-T1)+(T4-T3)+(T6-T5)\}+\{(T12-T11)+(T10-T9)+(T8-T7)\}}{2}$$

if $d < d_T^*$ *then*

$$\delta = \frac{\{(T2-T1)+(T4-T3)+(T6-T5)\}-\{(T12-T11)+(T10-T9)+(T8-T7)\}}{2}$$

else abort

Unlike SOM, both SDM and STM (proposed in the next section) require only neighboring nodes (*A* and *C*, *C* and *D*, *D* and *B*) to share pairwise secret keys. There is no need for *A* and *B* to share a secret key between them. To analyze this protocol, we first observe the communication between a pair of nodes, *A* and *C*, over the course of the protocol.

A (T1) → (T2)C ; C (T11) → (T12)A

These four timestamps are interrelated as follows:

$$T2 = T1 + \delta_{AC} + d_{AC}; T12 = T11 - \delta_{AC} + d_{AC} \ldots (4)$$

Note that we have added subscripts to both the offset and the end-to-end delay for clarity. Similarly, if you observe the communication between the other two node-pairs, *(C, D)* and *(D, B)*, the following relationships can be derived:

$$T4 = T3 + \delta_{CD} + d_{CD}; T10 = T9 - \delta_{CD} + d_{CD} \ldots (5)$$

$$T6 = T5 + \delta_{DB} + d_{DB}; T8 = T7 - \delta_{DB} + d_{DB} \ldots (6)$$

Adding the left hand and right hand side of equations (5) and (6), and introducing two new terms, δ_{AB} and d_{AB}, we can get the following two equations:

$$(T2-T1)+(T4-T3)+(T6-T5)=\delta_{AB}+d_{AB}\ldots(7)$$

$$(T12-T11)+(T10-T9)+(T8-T7)=-\delta_{AB}+d_{AB}\ldots(8)$$

Here, $\delta_{AB} = \delta_{AC} + \delta_{CD} + \delta_{DB}$ and $d_{AB} = d_{AC} + d_{CD} + d_{DB}$. The clock offset and the end-to-end delay can be calculated from equations (7) and (8).

4.2.1 Performance Evaluation

Just like SOM, the thresholding technique in SDM also requires the estimation of a new maximal delay d_T*. However, an important and crucial difference is that the end-to-end delay in SDM is not corrupted by the mac access delays; all the time stamps from *T1* to *T12* are below the mac layer. In fact, the end-to-end delay can be easily estimated knowing the number of hops. In the above example the end-to-end delay is equal to the cumulative sum of d_{AC}, d_{CD} and d_{DB}. All these three delays, d_{AC}, d_{CD} and d_{DB}, are equal to the end-to-end delay over a single link and hence, all of them follow the same Gaussian distribution calculated in Section 3.2.1, i.e., $d_{AC} = d_{CD} = d_{DB} \approx N(d_{avg}, \sigma)$. As a result, the end-to-end delay for SDM becomes a cumulative sum of independent Gaussian variables, i.e., $d_{AB} \approx N(nd_{avg}, \sigma\sqrt{n})$, where n is the number of hops. Given this, the maximal delay for an n hop network should be set to $d_T^* = nd_{avg} + 3\sigma\sqrt{n}$. The minimum synchronization precision and the maximum attacker impact will be given by $3\sigma\sqrt{n}$ and $6\sigma\sqrt{n}$ respectively. Note that σ is three orders of magnitude lower than σ_M for any value of n and hence, the minimum synchronization precision as well as the maximum attacker impact in SDM is much lower, around three orders of magnitude, than SOM.

However, the packet size of ack packets in SDM is larger than SOM resulting in an increased overhead. Every ack packet in SDM has to carry the state information (timestamps) about all the previous packets with it. For example, the acknowledgment packet from *C* to *A* contains 6 timestamps. In SOM, every acknowledgement packet has to just contain 2 timestamps, the same as SPS.

4.3 Secure Transitive Multi-hop (STM)

The STM protocol is executed as follows:

Secure Transitive Multihop Synchronization (STM)

1. A $\rightarrow$ C $\rightarrow$ D $\rightarrow$ B : A, B, N_A, sync
2. B : m1={B, D, notify}
   : M_1= MAC{K_{BD}}[B, D, N_A, m1]
   B $\rightarrow$ D: B, D, N_A, m1, M_1
3. D sync to B *(SPS)*
   D : m2={B, D, C, notify}
   : M_2= MAC{K_{BD}}[D, C, N_A, m2]
   D $\rightarrow$ C: D, C, N_A, m2, M_2
4. C sync to D *(SPS)*
   C : m3={B, D, C, A, notify}
   : M_3= MAC{K_{BD}}[C, A, N_A m3]
   C $\rightarrow$ A: C, A, N_A, m3, M_3
5. A sync to C *(SPS)*

Secure transitive synchronization scheme is essentially performed as SPS synchronization along all the links in the path from the source to the destination. The proper scheduling of node synchronizations is achieved in this protocol through an explicit notification by the receiver node to its upstream neighbor (sender node). Authentication is achieved by attaching a MAC at the end of this notification packet.

4.3.1 Performance Evaluation

Unlike SOM or SDM, STM does not require the estimation of any new maximal delay parameter. It runs the SPS protocol on every link and hence, the threshold verification gets divided into stages. Every link is evaluated separately using the same maximal delay, d^*, as in Section 3.2.1. This local verification has both advantages and disadvantages.

Imagine an external attacker which can carry out pulse-delay attacks on the link joining *C* and *D*). In both SOM and SDM, the thresholding verification is done only when the acknowledgement reaches back to node *A*. Thus, the system will detect the malicious attack only after running the complete protocol, which in our example pertains to transmission and reception of two packets per node. In STM, only nodes *C* and *D* will need to resynchronize when the thresholding verification at node *C* will fail. In fact, other nodes *A* and *B* won't even come to know about this malicious attack. Thereby, the overhead of countering a malicious attack in STM is lower than both SOM and SDM.

However, local verification gives extra freedom to the external attacker. It can introduce multiple pulse-delay attacks on every link simultaneously, thereby; the cumulative sum result of these attacks can be huge. Note that the pulse-delay factor introduced at every link can be at most 12σ as we run SPS on every link (refer Section 3.2.1). Thereby, the maximum attacker impact for an n hop network on STM can be $6\sigma n$, which is higher than the corresponding number for SDM, $6\sigma\sqrt{n}$. We note that both STM and SDM will achieve the same minimum synchronization precision, $3\sigma\sqrt{n}$.

Also, STM requires one extra packet transfer per node - the notification packet. In the absence of any malicious behavior, SOM and SDM require a total of $2n$ packet transmissions for synchronizing two nodes that are n hops away. STM requires $3n$ packet transmissions.

5. GROUP SYNCHRONIZATION

Several sensor network applications require all the nodes in a group to be time synchronized with each other. A few notable ones are – (1) Object tracking: The size, shape, direction, location, velocity, or acceleration of objects is determined by fusing proximity detections, done *at the same time*, from sensors at different locations, (2) Consistent state updates: The current state of an object is most accurately determined by the node that has seen the object *most recently*. This requires all the nodes in the cluster to have the same notion of time, (3) Distributed beamforming: Beam-forming arrays can perform "spatial filtering," receiving only signals arriving from a certain direction. This depends on the *relative time offsets* of the array's sensors, (4) Duplicate detection: The *time of an event* helps nodes in the cluster determine if they are seeing two distinct real-world events, or a single event seen from two vantage points. If they are indeed seeing the same event, they can further fuse their observations to get much meaningful information about the event. All these applications will function accurately only if the synchronization error between nodes in a group is bounded.

In the following sections, we propose two group synchronization protocols – Lightweight Secure Group Synchronization (L-SGS)

and Secure Group Synchronization (SGS). These protocols differ in their resilience to attacks and in the number of messages that the group nodes mutually exchange. Just like secure pairwise synchronization protocols, both L-SGS and SGS can detect the pulse-delay and packet modification attacks from external attackers. In addition, SGS can also detect the false timing reports by internal attackers but at the cost of a higher communication overhead.

5.1 System Model

In this model, we assume that group membership is known to all group nodes and that group members can authenticate each other using pairwise secret keys. We further assume that all group nodes reside in each other's power ranges; we do show that the proposed solutions can be easily extended to groups in which nodes are multiple hops apart. Given this assumption, we propose L-SGS and SGS exploiting the broadcast property of the wireless communication medium. Thereby, a message broadcast by a node in the group can be received by all the nodes in the cluster (here we assume that sensors have omnidirectional antennas). We represent the sending time of the packet at node i by T_i. T_{ij} represents the time at which the packet broadcast by node i is received at j. Notice that, these times are measured by two different clocks. T_i is measured in the local clock of node i (C_i) whereas T_{ij} is measured by the local clock of node j (C_j). We represent the offset (or the difference between the local clocks) between the two nodes by δ_{ij}. The delay for the packet transfer from i to j is represented by d_{ij}. Although this delay is the same for every pair of nodes, we add the subscripts for clarity.

5.2 Lightweight Secure Group Synchronization (L-SGS)

L-SGS is executed as follows:

Lightweight Secure Group Synchronization (L-SGS)

1. $G_1 \rightarrow *$: G_1, sync
2. $G_i(T_i) \rightarrow (T_{i1})G_1$: G_i, N_i
3. $G_1(T_1)$: m = $\{T_{i1},N_i,G_i\}^{i=2,...,N}$
   : M = $\{MAC\{K_{1i}\}[G_1, T_1,ack, T_{i1},N_i,G_i]\}^{i=2,...,N}$
   $G_1(T_1) \rightarrow (T_{1i})$*: G_1, T_1, ack, m, M
4. G_i : compute d=$((T_{i1}- T_i)+(T_{1i} - T_1))/2$
   If d≤d* *then* δ=$((T_{i1}- T_i)-(T_{1i} - T_1))/2$, *else* abort

In this protocol, one of the group members (G_1) initiates the time synchronization (step 1) to which the rest of the group members reply with messages containing their ids and challenge nonces (step 2). In step 3 of the protocol, G_1 replies with a single broadcast message to all other group nodes, containing MACs of the challenges and node ids. Note that the last protocol message (step 3) contains *N-1* triples $\{T_{i1}, N_i, G_i\}$, one for each G_i, containing the receipt time of the challenge packet from G_i (T_{i1}), the nonce of G_i and the node-id of G_i respectively. It also contains *N-1* MACs, one for each (G_1, G_i) pair, which enable each node G_i to authenticate the packet broadcast by G_1. In the last protocol step $G_2,...,G_N$ synchronize to G_1.

Note that this protocol does not need to be preceded by any explicit leader election algorithm. The initiating node G_1 does not need to be elected by other group members, but it can simply be the node that first broadcasts the sync packet after sensing a particular event. Therefore, L-SGS does not require any explicit scheduling in the group.

5.2.1 Performance Evaluation

We can observe L-SGS as an extension of the SPS protocol. In SPS, a single receiver synchronizes to a single sender; in L-SGS, multiple receivers synchronize to a single sender. L-SGS makes use of the broadcast property of the wireless channel. The total number of messages transmitted in an n-node cluster over the course of L-SGS is *n+1*. We note, however, that the last protocol message is significantly larger than other messages.

As L-SGS relies on the same primitives as SPS, the resistance of L-SGS to external attacks is the same as with SPS, which means that L-SGS is resilient to pulse-delay and message modification attacks. The minimum synchronization precision and the maximum attacker impact of L-SGS is the same as SPS, equal to 10μs (3σ) and 20μs (6σ) respectively.

However, L-SGS is not resilient to internal attacks. If node G_1 is malicious, it can produce a set of messages containing false times T_1 and T_{i1} for each node i and therefore have these nodes mutually desynchronized, while these nodes would believe to be synchronized.

5.2.2 Implementation

Step 3 of L-SGS requires G_1 to calculate *N-1* MACs on-the-fly. Clearly, this won't be feasible for large values of N. Given this, we briefly mention some of the alternative design options that we plan to explore in detail in the future. First, the broadcast message in Step 3 can be replaced by multiple unicast messages, one to every node in the cluster. This unicast message to node j will contain the MAC generated using the secret key shared between nodes G_1 and j. This will incur a significant communication overhead on G_1. Second, we can use public key signatures [14], which would make sure that a single signature (MAC) by node G_1 can be verified by all group nodes. This is assuming that all the nodes are aware of the public key of G_1, This will; however, incur a significant computational cost at G_1. Instead we can first establish a symmetric group key shared by all the nodes in the cluster. Node G_1 can then attach a single MAC in the ack packet, which is generated using this symmetric group key. This looks the most promising solution as the cost of establishing the symmetric group key can be amortized over multiple time synchronization events.

5.3 Secure Group Synchronization (SGS)

In this section, we propose a secure group synchronization protocol that also provides resiliency to internal adversaries; however, as we show, this protocol requires higher communication overhead than L-SGS. We start the section by proposing a simple consistency check that is the basic building block of the protocol.

5.3.1 Triangle Consistency

Consider a set of 3 nodes, *[i, j, k]*, that have successfully established pairwise offsets with each other. Imagine a triangle connecting these three nodes such that the link weight on every edge is equal to the offset between them. We refer to the traversal of such a triangle starting and ending at the same node as making a *cycle*. As we traverse in a cycle, we keep on accumulating the link weight. For example, a valid cycle starting from node i will be *[i->j, j->k, k->i]*. The final accumulated weight at the end of the

cycle will be $\delta_{ij} + \delta_{jk} + \delta_{ki}$. It can be easily observed that this should be equal to 0.

Corollary: If the result of any cycle is not 0, there exists an internal adversary in the set.

We note that the result of a cycle might not be exactly zero because of the drift and skew error. In practice, we will use a simple threshold based policy to account for this.

5.3.2 Protocol

SGS is executed as follows:

Secure Group Synchronisation (SGS)

1. $G_i(T_i) \rightarrow (T_{ij})^*$: G_i, N_i , sync; $(i=1,...,N), j \neq i$
2. $G_i(T'_i)$: $m=\{T_{ji}, N_j, G_j\}^{j=1,...,N; j\neq i}$
   : $M=\{MAC\{K_{ij}\}[G_i, T'_i, ack, T_{ji}, N_j, G_j]\}^{j=1,...,N; j\neq i}$
   $G_i(T'_i) \rightarrow (T'_{ij})^*$: G_i, T'_i, ack, m, M; $(i=1,...,N), j \neq i$
3. G_i : compute $\{d_{ij}=((T_{ij}- T_i)+(T'_{ji} – T'_j))/2\}^{j=1,...,N; j\neq i}$
   If all $d_{ij} \leq d^*$
   then $O_i=\{\delta_{ij}=((T_{ij}- T_i)-(T'_{ji} – T'_j))/2\}^{j=1,...,N; j\neq i}$
   else abort
4. G_i : $M=\{MAC\{K_{ij}\}[G_i, O_i]\}^{j=1,...,N; j\neq i}$
   $G_i \rightarrow *$: G_i, O_i, M; $(i=1,...,N), j \neq i$
5. G_i : check triangle consistencies
   : *if all* triangles are consistent,
   synchronize to the fastest clock

In this protocol, every group member (G_i) broadcasts a packet containing their ids and challenge nonces (step 1). In step 2 of the protocol, every member (G_i) broadcasts another packet containing the response to the challenges issued by all the other nodes in the first step. This packet contains *N-1* triples $\{T_{ji}, N_j, G_j\}$, one for each G_j, containing the receipt time of the challenge packet from G_j (T_{ji}), the nonce of G_j and the node-id of G_j respectively. It also contains *N-1* MACs, one for each (G_i, G_j) pair, which enables each receiver node G_j to authenticate the packet broadcasted by G_i in the second step. Note that the sending node G_i also includes the sending time (T'_i) in the response packet. The first two steps are reminiscent of sender-receiver synchronization; we are now establishing pairwise relationships between multiple senders and multiple receivers simultaneously using the broadcast property of the wireless communication medium. In the next step, each node G_i performs threshold verification on all the computed delays, d_{ij}, corresponding to the challenge-response with node G_j. This provides the resiliency to pulse-delay modifications from external attackers. If this step is successful, G_i will be able to construct a set O_i, containing clock offsets with other nodes in the group.

Step 4 and 5 provide the resiliency to SGS against internal attackers. As mentioned earlier, a simple triangle consistency check can be used to detect an internal attacker in a set of three nodes. Imagine a set of 3 nodes $[G_i, G_j, G_k]$. After step 3, G_i has the offsets with both G_j and G_k but it does not know the offset between G_j and G_k to perform the triangle consistency check. Step 4 of the protocol provides G_i with this information. In this step every node G_i broadcasts its offset set, O_i. This packet also contains *N-1* MACs, one corresponding to every other node in the group, so that the contents of the packet can be authenticated at every receiver node. Following this, each node performs multiple triangle consistency checks. If this step is successful, G_i increments its clock by the largest offset, $\max\{0, \max\{O_i\}\}$. As a result, G_i (and all the other nodes in the group) will get implicitly synchronized to the fastest clock in the group.

5.3.3 Performance Evaluation

The number of messages transmitted in an n-node cluster over the course of SGS is *3n*. SGS can also be viewed as an extension of the SPS protocol, whereby multiple nodes establish the pairwise clock relationships simultaneously. We note that the ack packet size in SGS (step 2) is significantly larger than other messages. Moreover, the computation of *N-1* on-the-fly MACs opens up the same design challenges as in L-SGS (refer Section 5.2.2).

As SGS is based on the same primitives as SPS, the minimum synchronization precision and the maximum attacker impact of SGS is the same as L-SGS and SPS, equal to 10µs (3σ) and 20µs (6σ) respectively. Just like L-SGS and SPS, it can counter external pulse-delay and message modification attacks. However, in addition SGS is resilient to internal attackers.

5.3.4 Resiliency to Internal Adversaries

In SGS, there is no fixed node to which the nodes in the group will synchronize. Synchronization is implicitly done to the fastest clock in the cluster. The identity of the node with the fastest clock does not get revealed till the end of the protocol. This removes the vulnerability of the protocol to a single point of failure such as G_1 in L-SGS. Furthermore, if an internal attacker tries to create inconsistencies in the running of the protocol, the triangle consistency check will fail and the process will be aborted. Let us illustrate this by an example.

Consider nodes *1-7* lying in the neighborhood of each other. Without any loss of generality, let us assume that node *1* is malicious and node *7* has the fastest clock. Node *1* can easily project itself as the fastest clock by creating a fictitious clock. At the end of the protocol, all the nodes will be synchronized to node *1* rather than node *7*. Note that this is not a valid attack. Although all the nodes are synchronized to some fictitious clock, they are consistently wrong. The relative error between them is still bounded.

Having realized this, the aim of node *1* will be to introduce inconsistencies in the system by portraying itself as the fastest clock to only a few subsets of nodes. Node *1* has full control over the values of the receipt times, T_{j1} for $j= \{2, 3, .., 7\}$, that it reports in the response packet (step 2). It can report arbitrary values creating inconsistent offset values at different nodes. We note that the thresholding verification in step 3 will provide some resiliency against arbitrary values reported by node *1* but it alone cannot guarantee safeguard against all possible malicious attacks.

For example, imagine the scenario where node *1* just targets node *2*. It reports a faulty value of the sending time T'_1 from the fictitious clock. It also reports the value of T_{21} from the same fictitious clock but reports an arbitrary value of receipt times for rest of the nodes, T_{j1} for $j= \{3, .., 7\}$. As a result, the thresholding mechanism on d_{j1} will fail at nodes $j=\{3,.., 7\}$ and hence, they will abort SGS. The thresholding mechanism will pass at node *2* and it will update its clock so that it synchronizes to the fictitious clock by node *1*. Thereby, in absence of steps 4 and 5, node *2* will be out of sync with rest of the nodes in the group. In SGS, when nodes exchange their offset sets, nodes *3-7* will not have valid pairwise offsets with node *1*. As a result, the triangle consistency check will fail at node *2*. Thereby, node *2* will also detect the

presence of malicious behavior in the system and will also abort SGS.

5.3.5 Extension to Multiple Hops

SGS is extendible to the scenario when group nodes are multiple hops away from each other. In this case, the steps *1* to *3* of the protocol will have to be changed. Currently we use the broadcast property of the wireless communication medium to establish pairwise offsets between the nodes in the group. If group nodes are multiple hops away, we would have to use any of the three protocols proposed in Section 3.3 for establishing these pairwise offsets. Steps 4 and 5 of the protocol will remain unchanged.

6. NETWORK SYNCHRONIZATION

Network-wide synchronization is achieved in two stages – (1) Hierarchical tree is established in the network with a reference node as the root, (2) Pairwise synchronization is performed along the edges of this tree using SPS. Every node synchronizes its clock to its parent in the tree. As a result, eventually all nodes in the network get synchronized to the reference node, which is the root of the hierarchical tree. Using SPS to perform pairwise synchronization provides resiliency against external attackers. However, the more challenging problem is to achieve secure network-wide synchronization when a few nodes in the network have been compromised. In this scenario, a compromised node can mislead all the nodes in its sub-tree to a different notion of time than the rest of the network.

In general, the problem of network-wide synchronization can be viewed as a composition of several multi-hop synchronizations between the reference node and rest of the nodes in the network. However the solutions for secure multi-hop synchronization rely on the assumption that all the intermediate nodes are trustworthy. Clearly, even one compromised node in the path can bring detrimental effects to the functionality of SDM or STM. A possible solution for countering compromised nodes is to use redundancy by using disjoint multiple paths to synchronize nodes. The protocols can be modified to be resistant to as many internal attackers as independent routes can be found. This will require that pair wise keys are established between all nodes in the network, or that some broadcast authentication mechanisms are in place, (e.g., Tesla[15]). To get an idea of how protocols can resist to internal attackers through independent routes, we refer the reader to a rich literature on security of routing protocols in multi-hop wireless networks [16], [17], [18]. We note that on a network-level this means that the reference node needs to maintain multiple trees simultaneously.

7. CONCLUSIONS

Existing solutions for time synchronization in sensor networks are not resilient to malicious behavior from external attackers or internally compromised nodes. We showcase the feasibility of a pulse-delay attack, whereby an attacker can introduce arbitrarily long delays in the packet propagation time directly affecting the achieved synchronization precision.

We then propose a suite of protocols for secure pairwise and group synchronization of nodes that lie in each other's power ranges and of nodes that are separated by multiple hops. Table 3 summarizes the key aspects of these protocols. These protocols offer different points of operation in the energy-accuracy subspace and the choice of the specific protocol should be made by the network designer depending on his application needs.

We believe that we have just scratched the surface in the solution space of secure time synchronization for sensor networks. Our future work includes investigation of secure network-wide synchronization schemes in the presence of multiple compromised nodes. In parallel, we are also developing better remedial actions against the malicious attacks than a simple abort of the protocol.

8. ACKNOWLEDGMENTS

This material is based on research supported in part by the Center for Embedded Networked Sensing (CENS), a NSF Science & Technology Centre, and by the Office of Naval Research (ONR) under the AINS Program. 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 CENS or the ONR.

Table 3: Summary of Secure Time Synchronization Protocols

Secure Synchronization	**Neighboring nodes**	**Multihop synchronization over n hops (Representative numbers for n=5)**			**Group synchronization of n nodes in a cluster**	
Protocols	SPS	SOM	SDM	STM	L-SGS	SGS
Minimum Synchronization Precision	3σ (≈10μs)	$3\sigma_M$ (≈25ms)	$3\sigma\sqrt{n}$ (≈25μs)	$3\sigma\sqrt{n}$ (≈25μs)	3σ (≈10μs)	3σ (≈10μs)
Maximum external attacker impact	6σ (≈20μs)	$6\sigma_M$ (≈50ms)	$6\sigma\sqrt{n}$ (≈50μs)	$6\sigma n$ (≈120μs)	6σ (≈20μs)	6σ (≈20μs)
Resiliency to internal attackers	-	Yes	No	No	No	Yes
Total number of transmitted messages	2	2n	2n	3n	n+1	3n
Ack Packet size[3]	-	Same	Large	Same	Large	Large

[3] Comparison with respect to the packet size in SPS.

9. REFERENCES

[1] Sundararaman, B., Buy, U., Kshemkalyani, D.. Clock synchronization for wireless sensor networks: A Survey. *Ad-hoc Networks,* 3(3): 281-323, May 2005.

[2] Elson, J., Girod, L., Estrin D.. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[3] Ganeriwal, S., Kumar, R., Srivastava, M. B.. Timing-sync protocol for sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys),* Los Angeles, CA, November 2003.

[4] Maroti, M., Kusy, B., Simon, G., Ledeczi, A.. The flooding time synchronization protocol. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys),* November 2004.

[5] Greumen. J. V., Rabaey, J.. Lightweight time synchronization for sensor networks. In *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA),* San Diego, CA, 2003.

[6] Sichitiu, M. L., Veerarittiphan, C.. Simple, accurate time synchronization for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.

[7] Zhu, S., Setia, S., Jajodia, S.. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the Tenth ACM Conference on Compute and Communications Security (CCS)*, Washington, D. C., October 2003.

[8] Eschenauer, L., Gligor, V. D.. A key-management scheme for distributed sensor networks. In *Proceedings of the Ninth ACM Conference on Compute and Communications Security (CCS)*, Washington, D. C., October 2002.

[9] Chan, H., Song, D., Perrig, A.. Random key predistribution scheme for sensor networks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[10] Xu, W., Trappe, W., Zhang, Y.. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the Sixth ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, 2005.

[11] Deb, B., Bhatnagar, S., Nath, B.. A topology discovery algorithm for sensor networks with applications to network management. In *Proceedings of the IEEE CAS Workshop,* September 2002.

[12] Deng, J., Han, R., Misra, S.. The performance evaluation of intusion-tolerant routing in wireless sensor networks. In *Proceedings of Information Processing in Sensor Networks (IPSN)*, April 2003.

[13] Titzer, B., Lee, D., Palsberg, J.. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the IPSN Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems (SPOTS)*, Los Angeles, CA, April 2005.

[14] Watro, R., Kong, D., Cuti, S., Gardiner, C., Lynn, C., Kruus, P.. TinyPK: Securing sensor networks with public key technology. In *Proceedings of the ACM Workshop on Security in Ad-hoc and Sensor Networks (SASN)*, Washington, D. C., October 2005.

[15] Perrig, A., Canetti, R., Song, D., Tygar, D. The TESLA broadcast authentication protocol. In *RSA Cryptobytes*, Summer 2002.

[16] Yaar, A., Perrig, A., Song, D.. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[17] Chun, Y., Perrig, A., Johnson, D.. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom)*, Atlanta, GA, September 2002.

[18] Papadimitratos, P., Haas, Z. J., Sirer, E.G.. Path Set Selection in Mobile Ad Hoc Networks. In *Proceedings of the Third ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, Lausanne, Switzerland, June 9-11, 2002.

[19] Karlof, C., Sastry, N., Wagner, D.. TinySec: A link layer security architecture for sensor networks. In *Proceedings of the Second ACM Conference on Embedded networked Sensor Systems (SenSys),* November 2004.

[20] Polastre, J., Szewczyk, R., Culler, D.. Telos: Enabling ultra-low power wireless research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks; Special track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS),* Los Angeles, CA, April 2005.