Secure Time Synchronization in Sensor Networks

SAURABH GANERIWAL
University of California, Los Angeles
CHRISTINA PÖPPER and SRDJAN ČAPKUN
ETH Zurich
and
MANI B. SRIVASTAVA
University of California, Los Angeles

Time synchronization is critical in sensor networks at many layers of their design. It enables better duty-cycling of the radio, accurate and secure localization, beamforming, and other collaborative signal processing tasks. These benefits make time-synchronization protocols a prime target of malicious adversaries who want to disrupt the normal operation of a sensor network. In this article, we analyze attacks on existing time synchronization protocols for wireless sensor networks and we propose a secure time synchronization toolbox to counter these attacks. This toolbox includes protocols for secure pairwise and group synchronization of nodes that either lie in the neighborhood of each other or are separated by multiple hops. We provide an in-depth analysis of the security and the energy overhead of the proposed protocols. The efficiency of these protocols has been tested through an experimental study on Mica2 motes.

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

General Terms: Algorithms, Design, Experimentation, Security

Additional Key Words and Phrases: Sensor networks, time synchronization, message authentication code, delay

ACM Reference Format:
Ganeriwal, S., Pöpper, C., Čapkun, S., and Srivastava, M. B. 2008. Secure time synchronization in sensor networks. ACM Trans. Inf. Syst. Secur. 11, 4, Article 23 (July 2008), 35 pages. DOI = 10.1145/1380564.1380571. http://doi.acm.org/10.1145/1380564.1380571.

Authors' addresses: S. Ganeriwal, Electrical Engineering Department, University of California, Los Angeles, CA 90095-1594; email: saurabh.ganeriwal@gmail.com; C. Pöpper, Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland; email: poepperc@inf.ethz.ch; S. Čapkun, Department of Computer Science, ETH Zurich, Switzerland; email: capkuns@inf.ethz.ch; M. B. Srivastava, Electrical Engineering Department, University of California, Los Angeles, CA 90095-1594; email: mbs@ucla.edu.

1. INTRODUCTION

Time synchronization is a critical middleware service in sensor networks, required at many layers of their design. Examples of existing sensor network applications where precise time is needed include consistently measuring the time of events detected by the sensors, 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 multisensor data, or coordinating on future action. Imagine the detrimental effect on the functionality of these applications if a malicious adversary is able to abuse the underlying time synchronization protocol; measured values will be associated with faulty time-stamps, 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, etc. It will be trivial for adversaries to perform replay attacks in security protocols that use time-stamping. Collaborative data processing and signal processing techniques will be adversely affected. Manzo et al. [2005] analyze the impact of malicious attacks on time synchronization to sensor network applications and middleware services such as localization (shooter localization [Maroti et al. 2004]), TDMA based channel sharing (flexible power scheduling [Hohlt et al. 2004] and Pedamacs [Ergen and Varaiya 2006]), and cryptography (μ-Tesla [Perrig et al. 2002; Schaller et al. 2007]). To quote from Manzo et al. [2005]: "In many application areas, time synchronization allows engineers to design simpler and more elegant algorithms. For certain classes of applications [...], algorithms cannot provide correct results without an accurate and reliable time-synchronization service."

Time synchronization has been thoroughly studied in sensor networks [Sundararaman et al. 2005] and there are several prototype implementations, such as RBS [Elson et al. 2002], TPSN [Ganeriwal et al. 2003], and FTSP [Maroti et al. 2004], which can achieve synchronization precision of a few microseconds. However, none of these protocols were designed to operate in adversarial settings. Realizing the inadequacy of existing time synchronization solutions, we have developed several schemes to achieve secure time synchronization in sensor networks. Our approach involves integrating security mechanisms into existing protocols as well as developing new protocols. We compare our schemes with recent proposals for secure time synchronization in sensor networks [Sun et al. 2005, 2006a]. Recently, a secure broadcast time synchronization scheme has been proposed [Rasmussen et al. 2007] that is not tailored for sensor networks.

Our contributions are multi-fold. First, we perform an in-depth security analysis of sender-receiver synchronization protocols [Ganeriwal et al. 2003]. 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. Specifically, we show the feasibility of a pulse-delay

attack, where an external attacker can arbitrarily skew the calculated clock offset between a pair of nodes. Examples of time synchronization protocols vulnerable to these attacks include TPSN [Ganeriwal et al. 2003], LTS [Greunen and Rabaey 2003], FTSP [Maroti et al. 2004], Mini/Tiny Sync [Sichitiu and Veerarittiphan 2003]. We note that these attacks are also feasible on receiver-receiver synchronization [Elson et al. 2002].

Second, we integrate security mechanisms into the basic approach of sender-receiver synchronization; we propose a protocol for secure pairwise time synchronization in sensor networks. We show that at a nominal overhead, our protocol can overcome attacks from external attackers. We further show that our protocol achieves the same synchronization precision as the insecure protocols achieve in a nonmalicious setting. In a malicious environment, our protocol can restrict the maximum impact of the attacker on the synchronization precision to under a few micro-seconds on Mica2 motes, and, therefore, can prevent the synchronization error from becoming unbounded. In other words, our protocol is resilient to attacks by sophisticated attackers (who, when our protocol is used, can skew the clock offset only by a few microseconds). Furthermore, we show that it is infeasible for a mote-class attacker to carry out successful attacks against our protocol.

Third, we propose a protocol for secure group synchronization. This protocol can be used by sensor network applications such as object tracking, intruder detection, beamforming and fire monitoring, where a primary requirement is that the synchronization error between any two nodes in a monitoring group is bounded. Our solution for secure group synchronization is not only resilient to external attacks, but it can also handle inconsistent behavior from a subset of nodes in the group. This behavior can either be the result of attacks performed by compromised nodes in the group or of nonmalicious corruptive processes such as hardware faults. Our protocol works in both scenarios, being completely agnostic to the cause of the inconsistent behavior. This protocol is based on the Byzantine agreement protocol, proposed in Lamport et al. [1982]. It does not require any explicit cooperation between the group nodes or any prior knowledge about the identity of internal attackers in the group. We will show the efficiency of our solution through simulations and through an experimental study performed on a group of Mica2 motes.

Finally, we discuss how protocols for secure pairwise and group synchronization can be extended to synchronize nodes multiple hops away from each other. We discuss how our protocols can be used to provide secure network-wide synchronization.

The organization of the article is as follows: In Section 2, we review some basics about sensor node clocks and time synchronization in sensor networks. Section 3 summarizes the attacker model used throughout the article. Sections 4 and 5.4.3 present solutions for secure pairwise and group synchronization of sensor nodes respectively. In Section 6, we present a protocol for securely synchronizing nodes multiple hops away from each other. We briefly discuss existing solutions for secure network-wide synchronization in Section 7 and conclude the article in Section 8.

2. TIME SYNCHRONIZATION IN SENSOR NETWORKS

Approaches for synchronizing a pair of nodes can be broadly classified as sender-receiver [Ganeriwal et al. 2003] or receiver-receiver [Elson et al. 2002]. In the sender-receiver protocol, a single node synchronizes its clock to the clock of a reference node using bidirectional communication. Receiver-receiver synchronization relies on measuring the differences in the reception times of signals sent by a reference node.

Protocols for network-wide clock synchronization [Elson et al. 2002; Ganeriwal et al. 2003] rely on these pairwise synchronization techniques to establish relationships between every network node and the reference node. This is typically achieved by forming paths in the sensor network, through which all network sensors synchronize to the reference nodes. Synchronization through multi-hop paths can be performed such that the first sensor on the path synchronizes to the base station and all other nodes synchronize pairwise to their preceding neighbors on the path in an ascending order [Ganeriwal et al. 2003] (i.e., besides routing the messages down the path all intermediate network nodes synchronize their own clocks).

In this work, we focus on the sender-receiver protocol, which is the basis for several synchronization schemes such as TPSN [Elson et al. 2002], LTS [Greunen and Rabaey 2003], and Mini/Tiny Sync [Sichitiu and Veerarittiphan 2003]. Although the malicious attacks on time synchronization that we discuss in the coming section are also applicable to receiver-receiver synchronization, the security mechanisms that we propose in this article are only applicable to sender-receiver synchronization.

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. The difference in the clocks of two sensor nodes 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), or (3) the frequency of the clocks can change differently over time because of aging or ambient conditions such as temperature (termed as the drift error). These errors can be summarized as follows:

1. Offset $\delta = C_A(t) - C_B(t)$
2. Skew $\eta = \frac{\partial C_A(t)}{\partial t} - \frac{\partial C_B(t)}{\partial t}$
3. Drift $\lambda = \frac{\partial^2 C_A(t)}{\partial t^2} - \frac{\partial^2 C_B(t)}{\partial t^2}$

In this article, we focus on achieving instantaneous synchronization between sensor nodes. Thus, the objective of our protocols is to estimate the offset error between the nodes, so that the clocks can be corrected accordingly.

We do not aim to estimate the drift or the skew errors. Thus, neither our attack model nor our security solutions concern protocols for long-term synchronization of sensor nodes such as RATS [Ganeriwal et al. 2005].

2.2 Sender-Receiver Synchronization

Pairwise sender-receiver synchronization is performed by a handshake protocol between a pair of nodes. This protocol is executed in three steps as follows:

Pairwise Sender-receiver Synchronization
1. A($T1$) $\rightarrow$ ($T2$)B: A, B, *sync*
2. B($T3$) $\rightarrow$ ($T4$)A: B, A, $T2$, $T3$, *ack*
3. A calculates offset $\delta = \frac{(T2-T1)-(T4-T3)}{2}$

Here, $T1$ and $T4$ represent times measured by the local clock C_A of node A. Similarly, $T2$ and $T3$ represent times 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 + δ + d. Here, δ and d represent the offset between the two nodes and the end-to-end delay respectively. At time $T3$, B sends back an acknowledgment 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 δ and the end-to-end delay d:

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

and, subsequently, synchronize its clock to B's clock: $C_A = C_A + \delta$.

3. ATTACKS ON TIME SYNCHRONIZATION

In this section, we review attacks on sender-receiver synchronization protocols. We start by presenting our system and attacker models.

3.1 System Model

Our system consists of a set of sensor nodes, which communicate using radio transmissions. We assume that the radio link between neighboring devices is bidirectional. The network is operated by an authority. The authority controls the network membership and assigns a unique identity to each node. Each pair of nodes holds a shared secret key that can either be manually preloaded into the nodes during the deployment phase or can be generated during the network setup phase using key establishment protocols (e.g., Perrig et al. 2001; Eschenauer and Gligor 2002; Chan et al. 2003; Liu and Ning 2003).

3.2 Attacker Model

We consider an omnipresent but computationally bounded adversary. She controls the communication channel in the sense that she is able to eavesdrop, insert, modify, and block arbitrary messages by adding her own signal to the channel (e.g., in order to jam the signal). Our attacker model is similar to the Dolev-Yao threat model [Dolev and Yao 1981] in that the attacker is limited by

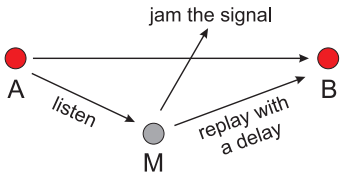


Fig. 1. Pulse-delay attack. The adversary (M) jams and replays messages at a later time, thus delaying their reception at the receiver.

the constraints of the used cryptographic methods regarding the contents of the messages that she can compose.

We distinguish two attacker models: internal and external. In the external attacker model we assume that none of the nodes involved in the protocol are compromised. Thus, an external attacker cannot authenticate herself as an honest network node to other network nodes or to the central authority. An internal attacker, however, controls one or more network nodes. We assume that when a node is compromised, its secret keys are known to the attacker. Subsequently, compromised nodes can authenticate themselves as legitimate nodes to the authority and to other network nodes. A noncompromised node can also misbehave because of nonmalicious corruptive processes such as software, hardware, or system faults. We classify these nodes likewise as internal attackers. As we will show, our protocols are indifferent to the cause of misbehavior.

3.3 External Attacks

External attacks are those in which an attacker manipulates the communication between pairs of mutually trusted nodes and causes them to desynchronize. We consider an attack as successful if the nodes calculate a faulty offset value, δ, but accept it as being correct.

There are several ways how the attacker can influence the calculation of the offset: (1) by modifying the values of $T2$ and $T3$, (2) by message forging and replay, and (3) by delaying the transmission of the messages between the nodes and thus increasing the value of $T2$ (or $T4$). The first two attacks can be easily prevented by traditional security primitives that protect the integrity and the authenticity of the transmitted messages.

The third attack, which we call the *pulse-delay attack*, is more challenging to detect. Notably, $T2$ is measured as the time at which the initial synchronization pulse packet sent by A is received at B. If an attacker delays the time at which B receives the synchronization pulse, she will be able to modify the computation of the offset at A. To delay the synchronization pulse an attacker can simply jam the initial pulse and replay it at some arbitrary time in the future (Figure 1) or, if the nodes are not within each others' transmission ranges, she can create a wormhole and then schedule the packets between the nodes at will. Both attacks cannot be prevented by the use of conventional cryptographic primitives. For the first scenario we assume that the attacker can jam the communication between two nodes by transmitting signals which disrupt

the packet reception. This can be achieved by using stealth and 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 well various types of jamming because of their low transmitting RF power (1mW), these sensor platforms are vulnerable to broadband jamming. Recently, [Xu et al. 2005] 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 $T4$. Note that if the nodes are not in each others' transmission ranges, the attacker does not need to perform jamming in order to achieve pulse delay, which makes pulse-delay attacks feasible even if sophisticated jamming detection techniques are deployed by the nodes.

If an attacker performs a pulse-delay attack on the transmitted messages of the time synchronization protocol, the clock offset δ and the end-to-end delay d become:

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

where Δ is the pulse delay introduced by the attacker.

This shows that the attacker can arbitrarily change the computed clock offset by varying the pulse delay. An important observation is that by performing a pulse-delay attack, the attacker also changes the computed end-to-end delay. Note that it is infeasible for the attacker to just change the computed clock offset δ without changing the computed end-to-end delay d. As we will show in later sections, we use this observation to detect pulse-delay attacks.

4. SECURE PAIRWISE SYNCHRONIZATION

To detect attacks on pairwise time synchronization we propose the following Secure Pairwise Synchronization Protocol (SPS). It is designed to be run by two nodes that reside within each others' communication ranges.

Secure Pairwise Synchronization (SPS)

1. A$(T1) \rightarrow (T2)$B: $A, B, N_A, sync$
2. B$(T3) \rightarrow (T4)$A: $B, A, N_A, T2, T3, ack,$
   $MAC_{K_{AB}}[B, A, N_A, T2, T3, ack]$
3. A calculates delay $d = \frac{(T2-T1)+(T4-T3)}{2}$
   If $d \leq d^*$ *then* $\delta = \frac{(T2-T1)-(T4-T3)}{2}$ *else* abort

In this protocol, message integrity and authenticity are ensured through the use of Message Authentication Codes (MAC) and of a key K_{AB} shared between A and B. This prevents external attackers from modifying values in the synchronization pulse or in the acknowledgement packet, without being detected. Furthermore, the attacker cannot impersonate node B as she does not know

the secret key K_{AB}. Replay attacks are prevented by using a random nonce N_A. In SPS, pulse-delay attacks are detected by comparing the computed message end-to-end delay d, with the maximal expected message delay d^*. If the attacker jams and replays the original packet, the new packet will arrive at the receiver with a delay that is the sum of the times that the attacker needs to receive the packet, process it, and transmit it, and of the synchronization delay Δ that the attacker tries to introduce. Note that the calculation of the end-to-end delay comes as an auxiliary benefit of the time synchronization protocol; we have not added any overhead on the functionality of sender-receiver synchronization. If the computed delay is greater than the maximal expected delay, we abort the offset calculation.

Clearly the performance of this scheme relies on the fact that the value of d^*, referred to as the *maximal delay*, is known. In the next section, we show that d^* can be accurately estimated, and that the packet transmission time is much longer than d^*, which enables the detection of the attack. We provide a testimony to this claim by carrying out experiments on Mica2 motes.

4.1 End-to-End Delay Estimation

There are three major components that contribute to the end-to-end delay of a packet traversing a wireless communication link. These are (1) medium access time: this time depends on the node density and the traffic patterns and can vary from few microseconds to several minutes, (2) packet transmission time: this is the time needed for a packet to be transmitted bit-by-bit by the radio of the sender node. This time is in the order of hundreds of microseconds, but is deterministic in nature. It depends on the packet size and on the sender's radio speed. (3) Signal propagation time: this is the time of the radio signal propagation in air and is in the order of a few nanoseconds. A detailed breakdown of the end-to-end packet delay can be found in Ganeriwal et al. [2003].

As can be observed from above, the medium access delay introduces the highest uncertainty in the end-to-end delay. A way around this problem is to time-stamp the packets below the MAC (Medium Access Control) layer. This approach has been used by existing time synchronization approaches in sensor networks [Ganeriwal et al. 2003] to achieve an accuracy of *few* microseconds.

4.1.1 *Measurements on Mica2 Motes.* In this section, we present the results that were obtained by a prototype implementation of SPS on Mica2 motes. The implementation uses the time-stamping library provided by TPSN [Ganeriwal et al. 2003] and the cryptographic library provided by TinySec [Karlof et al. 2004]. TPSN (Timing-sync Protocol for Sensor Networks) is one of the most prominent proposals for time synchronization in sensor networks that is based on sender-receiver synchronization; it uses MAC layer time stamping to achieve an accuracy of approximately 10μs for Mica2 motes. TinySec, a symmetric cryptographic library, is used to calculate the Message Authentication Code (MAC) on the fly. The packets are time-stamped as they are about to be transmitted at the physical layer.

One of our objectives was to gauge the distribution of the end-to-end delay using this implementation of SPS, so that an appropriate value of the maximal

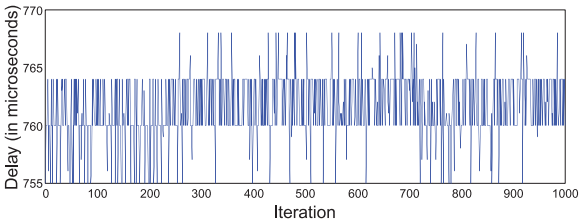


Fig. 2. End-to-end delay over a link.

Table I. Statistics of the End-to-End Delay Over a Link

Maximum delay	768 μs
Minimum delay	755 μs
Average delay d_{avg}	762 μs
Standard deviation	2.82 μs

delay d^* could be calculated. We ran this prototype implementation of SPS on a pair of motes and calculated the end-to-end delay (d in Equation (1)) for 200 independent runs. We repeated the complete procedure for 5 different pairs of motes in order to remove any hardware specific bias. In the end, we had 1000 independent measurements of the end-to-end delay. Figure 2 shows the actual delay measured in every run and Table I summarizes the statistics of the measurements. In our experiment, the measured delay concentrates on 14 different discrete values, each with a distance of about $1\mu s$. The reasons for the discrete results are twofold: (1) the time is measured with a limited granularity on Mica2 motes (about $0.25\mu s$) and (2) software delay happens at the granularity of the CPU clock cycles (with microsecond precision).

Under the assumption of a Gaussian distribution $d \sim N(d_{avg}, \sigma)$ as reported by authors in [Elson et al. 2002; Ganeriwal et al. 2003], the true delay will be in the interval $[d_{avg} - 3\sigma, d_{avg} + 3\sigma]$, with 99.7% confidence. We are time-stamping the first byte of the SFD (Start Frame Delimeter) at the transmitter and the end of the second byte of the SFD at the receiver. Thereby, we expect the calculated end-to-end delay to equal roughly two times the byte transmission time (the size of SFD is 2 bytes). We measured the byte transmission time to be roughly $380\mu s$. Note that the variable factor in time-stamping the packet is only the software uncertainty. The MAC uncertainty is removed by time-stamping the packet below the MAC layer and the signal propagation delay is of the order of nanoseconds (it takes 10 to hundreds of nanoseconds for a radio signal to travel a distance of 10 to 50 m).

Note that the average value of the end-to-end packet delay is not of much significance to us. The crucial factor is the standard deviation in the estimation of end-to-end delay. It is of the order of few microseconds and roughly 0.5% of the absolute value of d_{avg}. This implies that the end-to-end packet delay will be within a range of 3% of the average delay d_{avg}, with a probability of 99.9%. This allows us to choose an appropriate and stable value of the maximal expected delay d^*.

4.2 Performance Evaluation

In this section, we evaluate the performance of SPS using two metrics: (1) the synchronization precision which it can achieve in a non-malicious setting, and (2) the maximum impact of a pulse-delay attack on the achieved synchronization precision.

4.2.1 *Synchronization Precision in a Nonmalicious Setting.* SPS (and sender-receiver synchronization in general) involves two packet transfers; first from the sender to the receiver and then a reply back from the receiver. Since the node hardware and software are the same for all network nodes, we can model the end-to-end delay in each direction with the same Gaussian distribution, i.e., $d_{sender\_to\_receiver} = d_{receiver\_to\_sender} = d \sim N(d_{avg}, \sigma)$. If the end-to-end delay was the same in both communication directions, we would have been able to synchronize the nodes with zero error. From the above equation, it can be easily derived that $d_{variable}$, defined as the difference of the end-to-end delays in both directions, also follows a Gaussian distribution, i.e., $d_{variable} \sim N(0, \sigma\sqrt{2})$[1]. This variability in the delay directly contributes to the synchronization error. As shown in Elson et al. [2002] and Ganeriwal et al. [2003], the synchronization precision is related to the variability in the end-to-end delay as $\epsilon \sim d_{variable}/2$. Therefore, in this scenario the synchronization error also follows a Gaussian distribution given by $\epsilon \sim N(0, \sigma/\sqrt{2})$.

4.2.2 *Attacker Impact.* As shown in the previous section, with very high probability, the end-to-end delay will not exceed $d_{avg} + 3\sigma$. We can thus safely set the maximal delay to:

$$d^* = d_{avg} + 3\sigma \approx 771\mu s \tag{3}$$

We note that since the end-to-end delay follows a Gaussian distribution, it can take a very large value even in a non-malicious setting. By setting the maximal delay to $771\mu s$, our protocol will classify large delays as being the consequence of an attack. However, we would like to point out that this false negative probability is low ($\sim 0.15\%$).

The worst-case scenario (best case for the attacker) occurs, when the end-to-end delay in both directions, from sender to receiver and vice-versa, is equal to the minimum expected delay, $d_{avg} - 3\sigma$. In this case, the attacker can introduce a maximum pulse delay of $\Delta = 12\sigma$. The sender node will calculate the end-to-end delay as:

$$d = d_{avg} - 3\sigma + (12\sigma/2) = d_{avg} + 3\sigma \leq d^* \tag{4}$$

Thus, the maximum pulse delay that an attacker can introduce is 12σ (around $40\mu s$). The attacker will need to employ sufficiently fast and sophisticated hardware to carry out a pulse-delay attack that does not increase the

[1]Software delay occurs at two different nodes and, hence, can be considered independent. We assume that the signal propagation delay is independent in each direction. Even if it was dependent in the two directions, it would not have a significant impact on the distribution of the delay.

end-to-end delay by more than $40\mu s$. This is infeasible for an external mote-class attacker. The radio speed of Mica2 motes is 78.5 kbps. Even if the attacker receives and forwards the message byte-by-byte, the delay introduced by this operation will be at least $100\mu s$. For other existing sensor networking platforms with higher bitrates (in the order of few hundreds of kbps), the delay may be less, but the delay they introduce must still fall below a smaller maximal delay. Even if an external attacker, (e.g., with more sophisticated hardware), could tamper with the time synchronization by delaying messages, she could only do so within the precision of the synchronization precision—all other changes will be detected.

4.3 Pre-deployment Configuration of the Maximal Delay

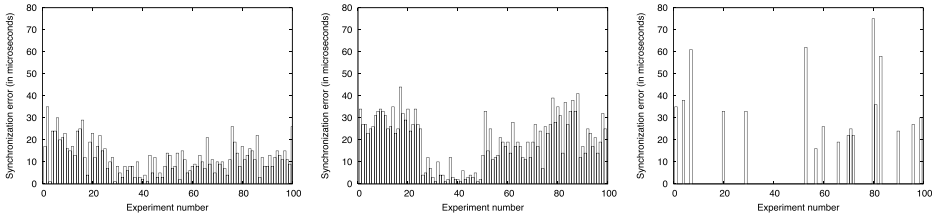
Interestingly, the distribution of the end-to-end delay, and hence the value of the maximal delay d^*, does not depend on the actual distance between the sensor nodes. The reasons for this are twofold: (1) The actual value of d^* is in the order of hundreds of microseconds, while the RF propagation delay over a wireless link (the only distance-dependent term in the packet propagation time from one node to another) is only in the order of nanoseconds for a distance of one meter. Most of the time is needed for transmitting the packet bit by bit at the physical layer, due to the relatively slow radios in these types of systems (maximum speed of 250 kbps). Therefore, even if a node is communicating with a nearby node (< 30cm) or a distant node (> 10m), the relative difference in the end-to-end delays for the two scenarios will be in the order of few nanoseconds. (2) Existing sensor nodes have clocks that can only measure to an accuracy of microseconds, making it infeasible to even calculate this difference.

We carried out an empirical evaluation of this assertion by measuring the end-to-end delay between pairs of nodes which were kept at different distances from one another. As anticipated, the distribution of the end-to-end delay was the same for all pairs. This has a strong implication: there is no need to estimate the value of the maximal delay d^* at runtime—it can be calculated before the deployment of the network and the nodes can be preconfigured with this value, greatly reducing the overhead. We do note that the value of the maximal delay will differ for sensor networking platforms that use different radios. For example, d^* will be different for Mica2 and MicaZ motes. However, a stable value of d^* can always be calculated, regardless of the sensor networking platform.

4.4 Resiliency to Attacks

The previous section provided a brief theoretical analysis of the performance of SPS in malicious and nonmalicious settings. In this section, we provide experimental results, obtained using Mica2 motes, in order to gauge both the synchronization precision achieved by SPS in nonmalicious settings and its resiliency against external attackers.

These experiments were done in the SOS [Han et al. 2005] environment, a reconfigurable operating system for sensor networks. The experimental setup consisted of three motes. One of the motes was designated as the sender node



(a) Nonmalicious setting (b) Pulse delay attack of 10 μs (c) Pulse delay attack of 25 μs

Fig. 3. Experimental study on Mica2 motes.

Table II. Statistics of the Synchronization Error in Malicious and Nonmalicious Settings. The Table Shows that the SPS Attack Detection Probability Increases with the Pulse Delay; When it Exceeds 30μs, the Attack is Always Detected

Experiment	Average error	Maximum error	Minimum error	Attack detection probability
Non Malicious	12.05 μs	35 μs	1 μs	NA
Δ = 10 μs	19.44 μs	44 μs	1 μs	1%
Δ = 20 μs	30.92 μs	61 μs	1 μs	37%
Δ = 25 μs	35.67 μs	75 μs	16 μs	82%
Δ = 30 μs	NA	NA	NA	100%

and it was responsible for initiating the SPS protocol. The other mote was designated as the receiver. A pairwise secret key was preconfigured into the motes, and was used to generate MACs for packet authentication. As mentioned in the previous section, the value of the maximal delay d^* was also preconfigured into the motes and it was set to 771μs. The maximum clock granularity of the local clocks at the motes was set to 1μs. After the completion of the SPS protocol, as indicated by the LEDs on the motes, a third mote was used to trigger an external interrupt at both motes. The synchronization error was calculated by the difference in the triggering time of the external interrupt at the two motes. The sender node added the newly calculated clock offset to its clock while reporting the time.

We ran this experiment 25 times with the same pair of motes. Then we repeated the whole experiment with three other pairs of motes to remove any hardware specific bias. Figure 3(a) plots the calculated synchronization error in these 100 independent runs and Table II summarizes the statistics. SPS achieves a synchronization precision of around 12μs. These numbers are similar to the ones achieved by existing (insecure) time synchronization protocols [Elson et al. 2002; Ganeriwal et al. 2003]. Thus, in a nonmalicious setting, SPS achieves the same accuracy as the existing protocols. The extra computation overhead of the MAC calculation, which is done on-the-fly, does not impact the accuracy of time synchronization.

Although there are implementations of jamming on Mica2 motes [Xu et al. 2005], it is infeasible to introduce a valid pulse-delay attack against SPS with a mote class attacker, as we explained in the previous section. Thereby, we decided to emulate an external attack, wherein we can introduce a pulse delay that is of the order of a few microseconds. Specifically, we modified the

underlying time-stamping library. In order to introduce a pulse delay Δ, we deliberately added Δ to the received timestamp $T4$ at the sender node. This is consistent with our attack model, as we do not modify any content in the packets.

Figures 3b and 3c plot the synchronization error between the two nodes for $\Delta = 10\mu s$ and $25\mu s$, respectively. The *zero* readings in the figures correspond to the scenario where the attack was detected, i.e., the calculated end-to-end delay exceeded the maximal delay and the protocol was aborted. Table II summarizes the statistics. The last column in this table marked as *Attack detection probability* refers to the percentage of cases when the attack was detected. We were able to introduce a pulse delay of $25\mu s$ with a nominal probability (≈ 0.2), but the attack was always detected with a pulse delay of $30\mu s$. This further justifies the need of sophisticated hardware to carry out a valid pulse-delay attack. As anticipated, the synchronization precision is inversely proportional to the magnitude of the attack (higher pulse delay). As shown in Table II, SPS can ensure that on average the two sensor nodes will never be desynchronized by more than $40\mu s$.

4.5 Recovery

In SPS, the process of time synchronization is aborted after the detection of a pulse-delay attack. Clearly, if an attacker keeps continuously jamming the communication channel, the sensor network won't be usable and therefore there is no point in developing secure time synchronization protocols for such scenarios. Instead, we focus on the scenario where the attacker intermittently jams the communication channel for a short duration of time. An interesting question is whether we can make our time synchronization protocol cognizant of the pulse-delay attack and develop remedial actions. Since the end-to-end delay is precisely known, we ask the following question: Is it feasible to estimate the pulse delay and recover the accurate clock offset between the two nodes from the wrongly calculated value?

In this section, we will show that it is infeasible to recover from a pulse-delay attack and the only solution is to rerun the protocol. Let us consider the best possible case, where the end-to-end delay is precisely known and is always equal to d_{avg}. A pulse-delay attack can be carried out in three different ways; we will consider each of these scenarios individually in the following sections.

4.5.1 *Scenario I: Pulse-Delay Attack on the Packet From A to B.* Let us consider the scenario where the packet from A to B is delayed by Δ [same as Equation (2)]. We introduce some more notations, after which, we can rewrite Equation (2) as:

$$d_{err} = \frac{(T2 - T1) + (T4 - T3) + \Delta}{2} = d_{avg} + \frac{\Delta}{2} \tag{5}$$

$$\delta_{err} = \frac{(T2 - T1) - (T4 - T3) + \Delta}{2} = \delta + \frac{\Delta}{2} \tag{6}$$

Here, δ_{err} and d_{err} represent the erroneous offset and the end-to-end delay respectively, as they are calculated by the sender node. Our objective is to calculate the right offset, represented by δ. Note that we have replaced the end-to-end delay d of Equation (2) by d_{avg}. The value of the pulse delay Δ can be calculated using Equation (5). Replacing this in the Equation (6), the offset becomes:

$$\delta = \delta_{err} - \frac{\Delta}{2} = \delta_{err} - (d_{err} - d_{avg}) \tag{7}$$

4.5.2 *Scenario II: Pulse-Delay Attack on the Packet From B to A.* Now we consider the case that the reply packet has been delayed by Δ. Equation (2) becomes

$$\begin{aligned} d_{err} &= \frac{(T2 - T1) + (T4 - T3) + \Delta}{2} = d_{avg} + \frac{\Delta}{2} \\ \delta_{err} &= \frac{(T2 - T1) - (T4 - T3) - \Delta}{2} = \delta - \frac{\Delta}{2} \end{aligned} \tag{8}$$

And the offset can be calculated as:

$$\delta = \delta_{err} + \frac{\Delta}{2} = \delta_{err} + (d_{err} - d_{avg}) \tag{9}$$

4.5.3 *Scenario III: Both Packets Are Under Attack.* In this case, imagine that the attacker carries out an attack on both packet transmissions and introduces a pulse delay of Δ_1 and Δ_2 respectively. Equation (2) becomes

$$\begin{aligned} d_{err} &= \frac{(T2 - T1) + (T4 - T3) + \Delta_1 + \Delta_2}{2} = d_{avg} + \frac{\Delta_1 + \Delta_2}{2} \\ \delta_{err} &= \frac{(T2 - T1) - (T4 - T3) + (\Delta_1 - \Delta_2)}{2} = \delta + \frac{\Delta_1 - \Delta_2}{2} \end{aligned} \tag{10}$$

Clearly, it is not possible to calculate the value of three variables, δ, Δ_1, and Δ_2, from just two equations. Furthermore, no additional packet exchanges between the two nodes can solve this problem, as every additional packet can also be potentially delayed by a pulse delay, Δ_3. As a result, every additional equation adds an additional variable Δ_3 to the system of equations, and therefore does not improve the relative difference between independent equations and the number of variables.

4.5.4 *Summary.* As can be seen above, it is theoretically feasible to recover the value of the offset in only two of the three possible scenarios. However, note that even these two scenarios have different recovery mechanisms; Equations (7) and (9). The paradox lies in the fact that it is impossible for a sender node

to differentiate between the three scenarios from the values of $T1$, $T2$, $T3$, and $T4$. Therefore, rather than wrongly calculating the clock offset, a sender node can only choose to abort and rerun the protocol, after having detected a pulse-delay attack.

4.6 Variation of SPS for Fast Hardware

Applying and implementing the SPS protocol on Mica2 motes has proven to be feasible and realistic. In particular, MACs can be computed on the fly and attached to a message while the message is being sent. This is necessary because the time-stamp inserted on the physical layer must indicate the exact transmission time of a certain bit in the message and the MAC can only be generated after the time-stamp has been inserted into the message. While Mica2 motes transmit with a 78.5 kbps bit rate, other existing types of motes can handle higher data transmission rates. This may render on-the-fly MAC computations infeasible, as was pointed out by Sun et al. [2006b] for the example of MicaZ motes (250 kbps bit rate). As a solution, the same authors propose to use prediction-based time-stamping, anticipating the (constant) delay to compute the MAC and adding this delay to the time-stamp.

Here, we propose two different approaches which eliminate the need for on-the-fly MAC computations. First, the original SPS protocol can be modified as follows: the MAC in message 2 does not contain any time-stamps (i.e., it is replaced by $MAC_{K_{AB}}[B, A, N_A, ack]$) while the MAC containing the time-stamps ($MAC_{K_{AB}}[B, A, N_A, T2, T3, ack]$) is sent in a third, delayed message. The MAC sent in the third message does not have to be computed on-the-fly, since its transmission and reception times are of no importance for the time synchronization. With this modification, we keep the properties of the original SPS protocol (message integrity, authenticity, and impeded impersonation as node B through the MAC and the shared key; detection of pulse-delay attacks). Only the time when A can verify the received time-stamps $T2$, $T3$ is delayed to the arrival of the third message.

Second, continuing on this idea, we propose a new protocol for secure pairwise synchronization, inspired by distance-bounding techniques [Brands and Chaum 1994]. In this protocol, the critical measurement of the message reception time performed by node A becomes independent of B's message containing the MAC. Consequently, the MAC does not have to be computed on the fly and, hence, does not impose any requirements on the transmission bit rate of the hardware. Our modified SPS protocol (E-SPS) is therefore hardware-independent.

Enhanced SPS (E-SPS)

1. A($T1$) $\rightarrow$ ($T2$)B: $A, B, N_A, sync$
2. B($T3$) $\rightarrow$ ($T4$)A: N_B where $T3 \geq T2$
3. B $\rightarrow$ A: $B, A, N_A, ack, T2, T3, N_B,$
   $MAC_{K_{AB}}[B, A, N_A, ack, T2, T3, N_B]$
4. A calculates delay $d = \frac{(T2-T1)+(T4-T3)}{2}$
   If $d \leq d^*$ *then* $\delta = \frac{(T2-T1)-(T4-T3)}{2}$ *else* abort

In our protocol, B's message that is used for the time measurement consists only of a fresh nonce N_B (2). N_B is then authenticated and integrity checked in a follow-up message (3) which may arrive with an arbitrary (though limited) delay. Since the content of message 2 does not depend on the time $T3$ when it was sent (as opposed to SPS), there are no requirements on the used hardware for sending message 2. Note that even though $T1 < T2 < T3 < T4$, A obtains the time-stamps in the following order: $T1$, $T4$, $T2/T3$. As in SPS, the integrity and authenticity of the messages are verified by using K_{AB} and a MAC. As before, successful replay attacks are precluded by the nonces N_A and N_B and pulse-delay attacks by the comparison with the maximal delay d^*. Preplay attacks, in which the attacker would send a message 2 before B's response, do not succeed because the attacker will be unable to provide a legitimate MAC for the preplayed nonce. Note that unlike in distance-bounding protocols, there is no need to use commitments in E-SPS, since the nodes mutually trust each other and share a secret key.

For the type of hardware we have been considering (Mica2), E-SPS achieves at least the same level of accuracy and end-to-end delay precision as the SPS protocol because both protocols agree in the structure of the synchronization, but the messages used in the time measurement of E-SPS are derived by faster operations.

5. GROUP SYNCHRONIZATION

The synchronization primitive in a sensor network should be able to provide additional services over and above the basic primitive of synchronizing a pair of neighboring nodes. Various sensor network applications require groups of nodes to agree on a common (e.g., global) time reference. Some notable applications are (1) Object tracking: The size, shape, direction, location, or velocity 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) 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. All these applications will function accurately only if the synchronization error between nodes in a group is bounded. Moreover, group synchronization can be equally beneficial for network-wide synchronization (see Section 7).

In this section, we present a protocol for secure time synchronization of a group of nodes, termed Secure Group Synchronization (SGS). We will show that SGS is resilient to both internal and external attackers. First, we present a simpler version of SGS that is only resilient to attacks from external adversaries. We have named this protocol Lightweight Secure Group Synchronization (L-SGS). As we will describe, L-SGS is not resilient to internal attacks resulting from compromised or faulty nodes. Afterwards, we propose a

mechanism which allows the synchronization of nodes even in the presence of internal attackers. That mechanism is based on the concepts of the Byzantine agreement [Lamport et al. 1982].

5.1 System Model

SGS is not bound to any specific group establishment or cryptographic protocol. Instead, the nodes establish memberships dynamically by running a secure membership protocol which initializes their membership statuses and maintains or updates them at runtime (changes may be due to lost and recovered nodes). This can be achieved by using a byzantine consensus protocol for group agreement, such as Berman and Garay [1993] or Garay and Moses [1998]. Note that nodes may be part of several groups in parallel since their goal is to synchronize on the same time reference. Although these protocols cause an overhead in terms of rounds and message sizes (e.g., polynomial-size messages and $m+1$ rounds to resist up to m byzantine nodes per group [Garay and Moses 1998]), they are executed only occasionally during the lifetime of the sensor network (e.g., when the topology changes). In other words, they occur with a frequency much below the synchronization frequency.

Based on the established groups, SGS is initiated repeatedly to reach and keep a uniform time reference. We will show that (1) any node in the group can initiate SGS whereupon all other nodes react and that (2) the end accuracy is independent of the order in which nodes send messages to each other. We assume that all group nodes are in a single broadcast domain (same neighborhood).

We require the following two properties from the underlying cryptographic library: (1) Every node in the group should be able to authenticate the messages received from other nodes, and (2) it is infeasible to impersonate and send valid messages on behalf of some other node. We achieve this by attaching message authentication codes to the packet that are generated using symmetric pairwise secret keys. As we will show, this requires the nodes to attach several MACs in every packet, one corresponding to every node in the group. Since we rely on on-the-fly MAC computations, we note that the protocols described below only work for small groups ($\sim$15 nodes). Although there is no logical bottleneck in the protocol that would prevent its scalability to hundreds of nodes, the computational complexity and time requirements make it impractical for large groups. In that case, an asymmetric cryptographic library or a modified algorithm following the idea indicated in Section 4.6 (i.e., delaying MACs to a separate message) would be better options.

5.1.1 *Notation.* We denote the number of nodes in the group by N and represent the sending time of the packet at node i by T_i. T_{ij} represents the time at which the packet broadcasted 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 follows the same distribution for every pair of nodes, we add the subscripts for clarity.

5.2 Lightweight Secure Group Synchronization (L-SGS)

In this section, we describe a lightweight secure group synchronization protocol.

Lightweight Secure Group Sync. (L-SGS) $\forall(i, j) \in (1, ..., N), j \neq i$:

1. $G_i(T_i) \rightarrow (T_{ij})* : G_i, N_i, sync$
2. $G_i(T'_i) : m = T_{ji}, N_j, G_j{}^{j=1,...,N;j\neq i}$
   $: M = MAC_{K_{ij}}[G_i, T'_i, ack, T_{ji}, N_j, G_j]^{j=1,...,N;j\neq i}$
   $G_i(T'_i) \rightarrow (T'_{ij})* : G_i, T'_i, ack, m, M$
3. G_i : compute $d_{ij} = ((T_{ij} - T_i) + (T'_{ji} - T'_j))/2$
   if $d_{ij} \leq d^*$ *then* $\delta_{ij} = \frac{1}{2}((T_{ij} - T_i) - (T'_{ji} - T'_j))$ *else* abort
4. G_i : Compute $C_{ij} = C_i + \delta_{ij}$
5. G_i : Compute $C^i_g = median(C_i, [C_{ij}]^{j=1,...,N;j\neq i})$

In this protocol, every group member broadcasts a packet containing its ID G_i and a challenge nonce N_i (step 1). This is called the *challenge packet* from G_i, which will be received by all neighboring nodes *. The exchange of these packets can be triggered by any node sending the first challenge, the other nodes in the group react in a random order. After the nodes have received $N_{min} \leq N-1$ replies of their group members (note that they might not get $N-1$ replies due to packet losses or compromised nodes not participating in step 1), the nodes begin with step 2 of the protocol, in which each member broadcasts a *response packet*. This packet contains up to $N-1$ triples (T_{ji}, N_j, G_j), one for each G_j involved in step 1. It contains the receipt time of the challenge packet from G_j (T_{ji}), the nonce of G_j (N_j), and the node ID (G_j) respectively. It also contains up to $N-1$ MACs, one for each pair (G_i, G_j), which allows each receiver node G_j to authenticate the response packet that was sent by G_i. In addition, G_i also adds the sending time T'_i of the response packet, both to the payload and to the MAC calculation. In step 3, each node G_i independently performs a threshold verification on each computed delay d_{ij}, corresponding to the challenge-response with node G_j. If the calculated end-to-end delay exceeds the maximal delay, we abort the offset calculation due to the pulse delay introduced by the external attacker. The first three steps are reminiscent of the sender-receiver synchronization; we establish pairwise relationships between multiple senders and multiple receivers simultaneously using the broadcast property of the wireless communication medium.

In step 4, each node G_i estimates the clock of all the other nodes in the group using pairwise offsets δ_{ij} and its local clock C_i. This estimation of the local clock of node G_j by node G_i is represented as C_{ij}. Finally, the group clock is calculated by taking the median of all these estimated local clocks and one's own local clock. We represent the estimation of the group clock by node G_i as C^i_g.

We demonstrate the efficacy of this procedure by a representative example. We consider a group of 4 nodes with the following local clocks: $C_1 = 10$, $C_2 = 20$, $C_3 = 30$ and $C_4 = 40$. We assume a nonmalicious environment so that node 1 calculates the clock offsets as: $\delta_{12} = 10$, $\delta_{13} = 20$ and $\delta_{14} = 30$. Using these pairwise offsets, node 1 will calculate the estimations of the local clocks as: $C_{12} = 20$, $C_{13} = 30$, $C_{14} = 40$, and finally the group clock as the median of $[10, 20, 30, 40]$, i.e., $C_g^1 = 25$. Similarly node 2 will calculate the offsets as: $\delta_{21} = -10$, $\delta_{23} = 10$ and $\delta_{24} = 20$, the local clocks as: $C_{21} = 10$, $C_{23} = 30$, $C_{24} = 40$, and the group clock as the median of $[10, 20, 30, 40]$, i.e., $C_g^2 = 25$. Note that $C_g^1 = C_g^2$.

5.2.1 *Complexity.* Each node transmits two packets in L-SGS – a challenge and a response packet. Hence, the total number of transmitted messages is $2N$. Albeit the response packet size is significantly larger than other messages, it has no impact on the accuracy of the time stamping as we are time-stamping the SFD bytes at both the transmitter and receiver. The only design challenge is the computation of $N-1$ on-the-fly MACs. As mentioned before, if N is large, the MACs can either be delayed (like in E-SPS) or a public key based cryptographic library can be used instead. In this latter case, each node attaches only one MAC in the response packet, which is generated using the private key of the node. All the receiver nodes can verify the authenticity of the packet by using the public key of the node.

5.2.2 *Synchronization Precision.* Following Section 4.2.1, the synchronization error in the estimation of the local clocks C_{ij} follows a Gaussian distribution given by $N(0, \sigma/\sqrt{2})$. Thus, the error in the estimation of the group clocks C_g^i also follows a Gaussian distribution given by $N(0, \sigma/\sqrt{2})$. The synchronization error ϵ_g is the difference in the estimation of the group clocks. Therefore, it follows a Gaussian distribution given by $N(0, \sigma)$. However, unlike SPS, L-SGS takes a considerable amount of time to finish; there are $2N$ messages being communicated on the channel. As a result, the offset between a node's clock at the end of the protocol might be different to the one at the beginning of the protocol because the node clocks may drift while SGS is running. To take this into account, we add an additional term to the synchronization error, $\epsilon_g \sim N(0, \sigma) + \theta(N)$. We note that the time needed to execute L-SGS, and hence the magnitude of $\theta(N)$, is directly proportional to the cardinality N of the group. Analyzing the behavior of $\theta(N)$ is a challenging problem and is beyond the scope of this article. In one of our previous research efforts [Ganeriwal et al. 2003], we calculated the average relative clock drift on Mica2 motes to be less than 1ppm, i.e., $1\mu s$ per second, which allows us to hypothesize that L-SGS will achieve a synchronization precision within a few microseconds for reasonable group sizes.

5.2.3 *Internal Attackers.* This lightweight version of SGS can successfully prevent pulse-delay attacks, using the same threshold mechanism that was described for SPS. However, it is not resilient to attacks from compromised nodes within the group. This can be seen by looking at the contents of the

response packet. The protocol relies on node G_i to accurately report the time T_{ji}, at which the challenge packet was received from node G_j in the first step of the protocol. There is no mechanism to cross-check the reported times and hence, a compromised node G_i can skew the offset δ_{ji}, calculated by a node G_j, by reporting an arbitrary value of T_{ji}. As a consequence, the local clock C_i of node G_i will be estimated differently by any node G_j and, hence, the medians calculated by the nodes will differ. We note that the attacker would also have to skew the value of T_i' accordingly, so that the calculated end-to-end delay is within the maximal delay. Even after altering a single clock offset, the internal attacker can make the synchronization error between the good nodes unbounded.

We would like to point out that this vulnerability of L-SGS to internal attackers is not an artifact of choosing the *median* as the aggregating function. Note that the median is more robust than a simple *average* aggregator and is in fact the best aggregator for an $\mathrm{R}^n \rightarrow \mathrm{R}$ aggregate. Instead of focusing our efforts on analyzing the tradeoffs associated with better aggregation modalities, we have developed a solution whose efficiency is not affected by the underlying aggregation modality.

5.3 Secure Group Synchronization (SGS)

In this section, we extend the lightweight secure group synchronization protocol to make it resilient to internal adversaries. L-SGS is not secure because nodes are not able to estimate the local clock of the internal adversary accurately. However, if a node does not cooperate, it is impossible for the rest of the group nodes to accurately estimate its local clock. An interesting point to note is that the group synchronization does not fail because nodes have a wrong estimate about the clock of a malicious node, but because their estimates are different. If we manage to ensure that the local clock estimates are *consistent*, we will be able to achieve accurate group synchronization. Based on this insight, we have developed a protocol that adapts the Byzantine agreement protocol proposed by Lamport in the context of clock synchronization [Lamport and Melliar-Smith 1985] and process synchronization [Lamport et al. 1982].

The modified protocol is executed as follows: Similar to the previous protocol, each node G_i calculates the pairwise offsets with all other nodes in the group and adds them to a set, termed as the *offset-set* O_i for node G_i. In the fourth step of the protocol, each node broadcasts its offset-set to all other nodes in the group. The integrity of this message is protected by attaching $N-1$ MACs, using the same mechanism as in the response packet. These MAC calculations do not have to be done on-the-fly as we are not time-stamping the packet. In the fifth step, each node runs the $SOM(\lfloor (N-1)/3 \rfloor)$ algorithm multiple times, once for every node G_j in the group, to calculate the estimation of the local node clocks, C_{ij}. This algorithm is based on the Byzantine agreement protocol, and we will explain it in detail in the next section. Finally, in step 6, each node calculates the group clock C_g^i by taking the median of the estimated local clocks. Essentially we have replaced the step 4 of the previous section

with two steps, 4 and 5. We will demonstrate how this makes SGS resilient to internal adversaries.

Secure Group Synchronization (SGS) $\forall(i, j) \in (1, ..., N), j \neq i$:

1. $G_i(T_i) \rightarrow (T_{ij})* : G_i, N_i, sync$
2. $G_i(T'_i) : m = T_{ji}, N_j, G_j{}^{j=1,...,N;j\neq i}$
   $: M = MAC_{K_{ij}}[G_i, T'_i, ack, T_{ji}, N_j, G_j]^{j=1,...,N;j\neq i}$
   $G_i(T'_i) \rightarrow (T'_{ij})* : G_i, T'_i, ack, m, M$
3. G_i : compute $d_{ij} = \frac{1}{2}((T_{ij} - T_i) + (T'_{ji} - T'_j))$
   if $d_{ij} \leq d^*$ then $\delta_{ij} = \frac{1}{2}((T_{ij} - T_i) - (T'_{ji} - T'_j))$ *else* abort
   $O_i = O_i \cup \delta_{ij}$
4. $G_i : M = MAC_{K_{ij}}[G_i, O_i]^{j=1,...,N;j\neq i}$
   $G_i \rightarrow * : G_i, O_i, M$
5. G_i : Run the SOM($\lfloor (N-1)/3 \rfloor$) algorithm to compute C_{ij}
6. G_i : Compute $C^i_g = median(C_i, [C_{ij}]^{j=1,...,N;j\neq i})$

5.3.1 *Byzantine Agreement Algorithm.* The SOM algorithm is based on the OM [Lamport et al. 1982] and the COM algorithm [Lamport and Melliar-Smith 1985] in the context of process and clock synchronization respectively. SOM, which stands for *Secure time synchronization OM*, is a modified version that suits our system settings. While both OM and COM involve multiple rounds of message exchanges between the participating nodes, in SOM we have replaced the multiple rounds of message exchanges by multiple rounds of computations at every node. Note that communicating 1 bit consumes around the same amount of energy that is needed to execute 1000 computations at a typical sensor node. Thereby, this design choice is apt for sensor networking systems.

SOM is a recursive algorithm that involves multiple rounds to compute C_{ij}. In SOM, each node uses other group members to compute C_{ij}. Therefore, we add additional indexes to the notation for clarity. We use $C^{k,r}_{ij}$ to represent the local clock of node G_j estimated by node G_i using the node G_k in the group after the r^{th} round of running the SOM algorithm. Clearly, G_k cannot be equal to G_j, although it can be equal to G_i. Further, we use the parameter m to represent the maximal number of internal attackers in the group. The pseudo code for the SOM algorithm run by node G_i is as follows:

SOM(m) to estimate C_{ij}:
  C_{ij} = median[(N-1) executions of SOM(m) to estimate $C^{k,m}_{ij}$ |
    $(i, j, k) \in (1, ..., N), k \neq j, j \neq i$]
SOM(1) to estimate $C^{k,1}_{ij}$:
  $C^{k,1}_{ij} = C_i + \delta_{ik} + \delta_{kj}$
SOM(m) to estimate $C^{k,m}_{ij}$ for $m > 1$:
  $C^{k,m}_{ij} = \delta_{kj}$ + median[(N-2) executions of SOM(m-1) to estimate $C^{t,m-1}_{ik}$ |
    $(i, j, k, t) \in (1, ..., N), t \neq k \neq j, j \neq i$]

In the following, we demonstrate the SOM algorithm by means of examples and then analyze its properties in terms of resilience to attackers, complexity, and synchronization precision.

5.3.2 *Example.* We reconsider the same representative example of 4 nodes with local clocks $C_i = 10i$. Assume that node 4 is malicious ($m = 1$) and that it wrongly propagates the receive times in the response packet, so that other nodes in the group calculate the offsets as: $\delta_{14} = \alpha$, $\delta_{24} = \beta$, and $\delta_{34} = \gamma$. All the other clock offsets are calculated accurately.

We go through the SOM algorithm as executed by nodes 1 and 2 to estimate C_{14} and C_{24}, respectively. They compute $C_{14} = median[C_{14}^{1,1}, C_{14}^{2,1}, C_{14}^{3,1}]$ and $C_{24} = median[C_{24}^{1,1}, C_{24}^{2,1}, C_{24}^{3,1}]$. By running SOM, the following values will be calculated:

$$\begin{array}{ll} C_{14}^{1,1} = C_1 + \delta_{11} + \delta_{14} = 10 + 0 + \alpha & C_{24}^{1,1} = C_2 + \delta_{21} + \delta_{14} = 20 + (-10) + \alpha \\ C_{14}^{2,1} = C_1 + \delta_{12} + \delta_{24} = 10 + 10 + \beta & C_{24}^{2,1} = C_2 + \delta_{22} + \delta_{24} = 20 + 0 + \beta \\ C_{14}^{3,1} = C_1 + \delta_{13} + \delta_{34} = 10 + 20 + \gamma & C_{24}^{3,1} = C_2 + \delta_{23} + \delta_{34} = 20 + 10 + \gamma \\ C_{14} = median(10 + \alpha, 20 + \beta, 30 + \gamma) & C_{24} = median(10 + \alpha, 20 + \beta, 30 + \gamma) \end{array}$$

As can be observed, $C_{14} = C_{24}$ and hence, both nodes 1 and 2 reach a consistent estimate about the local clock of node 4. We note that C_{*4} does not reflect the right clock of node 4. Our aim is not to accurately estimate the clock of node 4, but to ensure that the honest nodes in the group reach a consensus. For completion, we note down the other local clock estimates of node 1 and node 2:

$$\begin{array}{ll} C_{12} = median(20, 20, 10 + \alpha - \beta) = 20; & C_{13} = median(30, 30, 10 + \alpha - \gamma) = 30 \\ C_{21} = median(10, 10, 20 + \beta - \alpha) = 10; & C_{23} = median(30, 30, 20 + \beta - \gamma) = 30 \end{array}$$

As can be seen, all the local estimate clocks of node 1 and node 2 are the same. Also, $C_1 = C_{21}$ and $C_{12} = C_2$. Therefore, $C_g^1 = C_g^2$, and hence nodes 1 and 2 are able to derive a common group clock, even in the presence of an internal adversary.

5.3.3 *Recursion.* In order to understand the need for a recursive algorithm, we consider a representative example of 7 nodes, where nodes 6 and 7 have been compromised. We will show that nodes 1 and 2 will not be able to reach a consistent estimate of C_{17} and C_{27} by running the SOM(1) algorithm. However, the recursive algorithm, SOM(2) will be able to remove this discrepancy.

Assume that node 6 and 7 wrongly propagate the reception times in the response packet, so that nodes 1 and 2 calculate the offsets as: $\delta_{16} = \alpha_6$, $\delta_{17} = \alpha_7$, $\delta_{26} = \beta_6$, and $\delta_{27} = \beta_7$. We now demonstrate the inaccuracy of the SOM(1) algorithm:

$$\begin{array}{c} C_{17} = median[C_{17}^{1,1} \ldots\ldots\ldots, C_{17}^{6,1}]; \quad C_{27} = median[C_{27}^{1,1} \ldots\ldots\ldots, C_{27}^{6,1}] \\ C_{17}^{6,1} = C_1 + \delta_{16} + \delta_{67} = 10 + \alpha_6 + \delta_{67}; \quad C_{27}^{6,1} = C_2 + \delta_{26} + \delta_{67} = 20 + \beta_6 + \delta_{67} \end{array}$$

As can be noted, $C_{17}^{6,1} \neq C_{27}^{6,1}$ and, hence, C_{17} and C_{27} may be unequal. Similarly, the nodes will have inconsistent estimates of $C_{i6}^{7,1}$ and, hence, of C_{i6}.

The recursive version of SOM allows us to remove this inconsistency by first making nodes 1 to 5 reach a consistent estimate on $C_{i7}^{6,1}$ and, then making them use it to estimate C_{i7}.

5.3.4 *Internal Attackers.* SOM requires the parameter m, the number of internal attackers in the group, as input. However, a tricky question is how to estimate this value of m? Note that the identities or number of faulty nodes are not known before the execution of the algorithm. We only know the cardinality N of the group. In absence of any prior knowledge and to avoid overloading the system with any detection protocol, we devise a strategy to handle the worst-case scenario. We choose m to be equal to $\lfloor (N-1)/3 \rfloor$, the maximal number of attackers accepted by OM(m) and COM(m) in [Lamport et al. 1982]. SOM($\lfloor (N-1)/3 \rfloor$) will be able to handle at most $\lfloor (N-1)/3 \rfloor$ adversaries. If the number of internal attackers is bigger, no version of SOM can safeguard SGS and hence, the choice of m is a nonissue. If the number of internal attackers is smaller, we will be unnecessarily running some extra rounds of computation during the execution of SOM. However, since over-execution of SOM does not affect its accuracy, this choice of m is still justified.

THEOREM 5.1. *For any m, SOM(m) satisfies a correct time agreement among all noncompromised nodes if more than 3m nodes take part in the protocol and at most m out of them are faulty.*

PROOF. We develop a proof for this theorem based on the proofs of OM and COM [Lamport and Melliar-Smith 1985]. We start with the base case of SOM(1) and then develop the proof using induction on m. Assuming there is one malicious node ($m = 1$), the number of honest nodes needs to be greater than $3m$, i.e., at least 4, for the protocol to yield correct node synchronization. Let us take the worst-case scenario of 4 nodes. The malicious node can at most affect 1 out of 3 possible values of $C_{ij}^{k,1}$ for a node j, calculated by node i. Since the majority of the values are correct, a correct median will be calculated.

Now we assume that SOM($m-1$) is correct for $m-1$ and show that it remains correct for m, where $m < \frac{N}{3}$. The recursion causes SOM(m) to run $(N-1)$ executions of SOM(m) to estimate $C_{ij}^{k,m}$ where each run invokes $(N-2)$ executions of SOM(m) to estimate $C_{ik}^{t,m-1}$. Since the algorithm is by assumption correct for SOM($m-1$) and all correct nodes hold the same vector of values for all other honest nodes (though they do not know which ones are honest), the median of the values $C_{ik}^{t,m-1}$ will be correct. Then, on the upper recursion depth, out of the $(N-1)$ values $C_{ij}^{k,m}$, at most m will be incorrect, namely the ones where node k is corrupted. Note that we only allow m nodes to be malicious. This means a correct median will be calculated if the majority of all received values are correct, i.e., $\frac{N-1}{3} \geq m$ or $\frac{N}{3} > m$. □

5.3.5 *Complexity.* Each node transmits three packets in SGS—a challenge packet, a response packet and finally a packet containing its offset-set. Therefore, the total number of messages transmitted in SGS is $3N$. SOM(m) needs m iterations to remove the effect of m malicious nodes. We note that the number of computations needed are not linear but exponential in the number of rounds. The recursion terminates at the k^{th} round, when $m - k = 1$. Therefore, the total number of computations are of the order of N^m. In our case, we fix

m to be equal to $\lfloor (N-1)/3 \rfloor$ and hence, the computation complexity of SGS is $O(N^{\lfloor (N-1)/3 \rfloor})$.

5.3.6 *Synchronization Precision.* In Lamport and Melliar-Smith [1985], Lamport proves that the maximum synchronization error of the COM(m) algorithm is always bounded by $(6m + 4)\epsilon$. In his notation, ϵ refers to the accuracy with which pairwise nodes can derive the clock offsets. In our scenario, we have shown that this error follows a Gaussian distribution given by $N(0, \sigma)$. There exists no analysis of the average performance of the COM(m) or the OM(m) algorithm or on the conditions in which this worst-case scenario is attained, which can be mainly attributed to the complexity of the COM and OM algorithms. In order to do a first-order analysis, we assume that the average error is equal to the maximum error and hence, the synchronization precision of SGS follows a Gaussian distribution given by $N(0, \sigma(6\lfloor (N-1)/3 \rfloor + 4))$. Finally, we add a term to the error in order to take into account the drift of the clocks while running the SGS algorithm. Therefore, the synchronization error is given by $\epsilon_g \sim N(0, \sigma(6\lfloor (N-1)/3 \rfloor + 4)) + \theta(N)$.

5.3.7 *Maximum Attacker Impact.* The maximum attacker impact is also bounded by $(6\lfloor (N-1)/3 \rfloor + 4)\epsilon$, where ϵ refers to the maximum attacker impact on the derived pairwise clock offsets among the honest nodes in the group. As derived in the previous section, the maximum pulse delay that can be introduced by an attacker is 12σ, and hence the maximum clock difference that can be introduced by the attacker is equal to $(6\lfloor (N-1)/3 \rfloor + 4)12\sigma$.

5.3.8 *Related Work.* Group synchronization has been a widely researched problem in the traditional computing domain. Although Lamport started it with the COM algorithm [Lamport and Melliar-Smith 1985], several results have been published that can achieve better results than the COM algorithm on the metrics of error or complexity. In Lundelius and Lynch [1984], the COM algorithm is extended to achieve a synchronization precision that is independent of the number of processes involved. In Srikant and Toueg [1987], the authors propose an algorithm that does not only achieve better accuracy than the COM algorithm but also comes close to achieving optimal synchronization. Both these algorithms are based on one-way broadcast primitives where a group node starts the algorithm by broadcasting a message. After hearing this message, all the other nodes broadcast a response. After a few more rounds of message exchanges the nodes reach a consensus.

The problem with this class of algorithms, which are based on one-way broadcast primitives, lies in their incapability of handling pulse-delay attacks. Typically, the efficacy of these algorithms is proven (as in Lundelius and Lynch [1984] and Srikant and Toueg [1987]) by assuming that the time for a synchronization round is bounded, i.e., the message broadcasted by a node will reach all receivers within a fixed amount of time. Although this is valid for messages exchanged by two nodes in a wired or a wireless nonmalicious setting, it is not valid in environments, where an external attacker can increase the reception time of the message at selected nodes (pulse-delay attack). As a result,

different nodes get an inconsistent notion of the *round* of the algorithm, which will result in an inaccurate estimation of the group clock.

Several other algorithms, such as Halpern et al. [1984]; Mahaney and Schneider [1985]; Dolev et al. [1986]; Sun et al. [2005; 2006a], have focused on the problem of periodic resynchronization of group nodes. The problem is formulated as one of deciding the time interval ζ, and the algorithm that is run periodically for resynchronizing a group of nodes within a specified precision ϵ, given the fact that the nodes start off with clocks that are roughly synchronized within ϵ. This assumption is not valid for our system model, where we try to achieve synchronization between clocks that might be arbitrarily apart from one another at the onset of the algorithm. Besides, our objective is not to devise a periodic resynchronization algorithm but to achieve instantaneous group consensus between participating nodes.

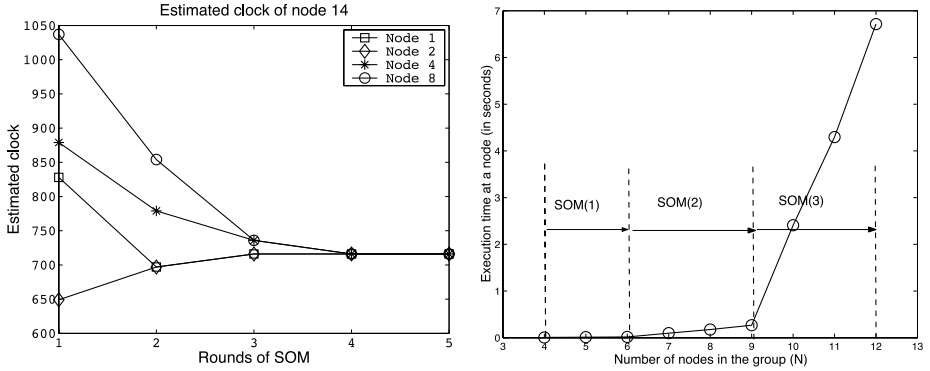
5.4 Performance Evaluation

In this section, we will first demonstrate the efficacy of SGS through simulations. We will then show the feasibility of implementing SGS on sensor networking systems by providing a prototype implementation on Mica2 motes. We will analyze the time requirements needed to execute SGS and report synchronization error results that were obtained by running real experiments on a group of motes.

5.4.1 *Simulation Study.* Our first objective was to verify the efficacy of the SOM algorithm in preventing attacks from internal adversaries. For this we implemented the algorithm in the *C* programming language and tested its efficacy on a desktop computer. We considered a system of 14 nodes where the local clock of node i was set to $C_i = i \cdot 10$. Nodes numbered 11 to 14 were modeled as adversarial nodes. The internal attack was modeled by adding a random value to the correct offset between a good and a compromised node as follows ($i \in (1, ..., 10)$, $j \in (11, 14)$):

$$\delta_{ij} += (5 \cdot i) + (10 \cdot j) + random\,[0, 1000].$$

Figure 4a shows the value of the local clock of the compromised node 14 as estimated by the nodes 1, 2, 4, and 8 after each iteration of SOM. As can be observed from the figure, the nodes have completely different views of the local clock C_{14} at the beginning, but they are able to reach a consensus after four iterations. Note that we ran the SOM algorithm independently for every node. Thus, without any explicit cooperation or message exchange, the nodes were able to reach a consensus about C_{14}. According to our assertion and since there are 14 nodes ($N = 14$), the nodes will require at least 4 rounds ($m = \lfloor (N-1)/3 \rfloor$) to reach a consensus. This is indeed the case, as can be observed from Figure 4a. Any fewer rounds of SOM (<4) are inadequate. Although a subset of nodes can reach a consensus (nodes [1, 2] and [4, 8] agree after three rounds of SOM), a complete consensus can only be guaranteed after running the complete four rounds of SOM. Furthermore, overrunning SOM (fifth round) does not have any impact on its accuracy.



(a) Performance of SOM in simulations. (b) Time complexity of SGS on motes.

Fig. 4. Simulation of SOM 4(a) and implementation of SGS on Mica2 motes 4(b).

We ran the implementation of SOM for several different values of N, varying from 4 to 30. We have not come across a single experimental run in which there was a discrepancy between the group clocks derived by the good nodes, provided that the number of adversarial nodes were less than one-third of the group strength.

5.4.2 *Time Complexity on Motes.* In this section and the next, we discuss the experimental results that were obtained from the prototype implementation of SGS (including SOM) on Mica2 motes. This implementation was done under the SOS environment. Our first objective was to profile the time needed by a mote to execute the SOM algorithm. This was done using Avrora [Titzer et al. 2005]. Avrora provides a cycle-accurate simulation of the AVR microcontroller, allowing real programs to be executed with precise timing. In addition, it provides analysis and profiling tools to gauge the cycle count of the programs. Avrora's most interesting feature is that the same embedded code running on Mica2 motes can be directly fed into the simulator, thus eliminating experimental errors introduced in moving from real-world settings to simulations.

Figure 4b plots the time taken by a mote to run the SOM algorithm with respect to the group strength (N). Note that m is always set to $\lfloor (N-1)/3 \rfloor$. Avrora provides a flexible framework to set up profile points at any place in the code. Using this profile points, the accurate cycle count can be derived for any snippet of code. In our scenario, the profile points were set to the beginning and the end of the SOM algorithm. Afterwards, the time measurement was derived by multiplying this cycle count with the frequency at which the atmega128 microcontroller runs in the Mica2 motes ($\sim$ 8MHz). As can be observed from Figure 4b, SOM takes less than a second to run in scenarios where the group size is below 10. Even for higher group sizes, SOM takes few seconds ($<$ 10 s) to run. A keen observation regards the nature of the graph which can be best modeled as piecewise linear, with transition points at $(N-1)$ mod 3 = 0, i.e., 4, 7, 10, etc. At these transition points, the number of SOM rounds increases.

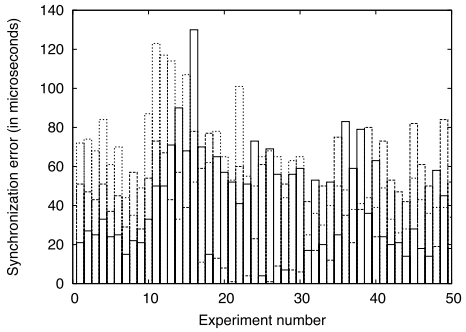


Fig. 5. Performance of SGS on motes.

Table III. Statistics of the Synchronization Error in SGS

Average error	47.13 μs
Maximum error	130 μs
Minimum error	1 μs

5.4.3 *Synchronization Precision on Motes.* We next carried out a set of experiments on Mica2 motes to gauge the synchronization error of SGS. The experimental setup consists of five motes, numbered 1 to 5. In practice, SGS can be started by any random node, following a group establishment. In our scenario, we designated node 1 to be the initiator at all times for ease of experimentation. Each node ran the SGS algorithm independently. Thus, it first broadcasts the challenge packet, waits for some time to receive the challenge packet from all the other nodes, broadcasts the response packet, waits to receive the response packet from all the other nodes, calculates the pairwise offsets and then broadcasts its offset-set. Thereafter, each node executes the SOM algorithm independently to derive the group clock.

We configured mote 4 to be the internal attacker. Its time-stamping library was modified to report arbitrary time stamps, resulting in an inaccurate and inconsistent calculation of the pairwise offset by the other nodes. After the completion of the SGS algorithm, as indicated by the LEDs at the motes, mote 5 was used to trigger an external interrupt. The three remaining motes reported the triggering time of this external interrupt, i.e., added the clock offset with the group clock to its local clocks, resulting in times C_g^1, C_g^2, and C_g^3. The synchronization error was calculated by the difference in these reported times.

We conducted 10 independent runs with a given experimental setup. Note that each experiment gives us three readings of the synchronization error, $\|C_g^1 - C_g^2\|$, $\|C_g^2 - C_g^3\|$ and $\|C_g^1 - C_g^3\|$. We repeated the whole experiment with five different setups, either by using new motes or by changing the identity of the adversarial nodes. Figure 5 plots the synchronization error in these 50 independent runs, each of the three fractioned bars representing one error reading. Table III summarizes the statistics. Even in the presence of an internal adversary, SGS was able to achieve an average synchronization precision of around $50\mu s$.

6. SECURE PAIRWISE MULTI-HOP SYNCHRONIZATION

In the above sections, we have established clock offset relationships between nodes that are in each other's communication range. In this section, we present protocols that enable distant nodes, multiple hops away from each other, to establish pairwise clock offsets. We assume that two sensors can obtain sets of communication paths between each other through a priori knowledge of the network topology. This information can be obtained either by topology discovery [Deb et al. 2002] or by routing information [Deng et al. 2003]. The precision achieved by the multi-hop protocols depends on the accuracy of the topology information.

6.1 Secure Simple Multi-hop Synchronization (SSM)

The SSM protocol is a simple extension of the SPS protocol, in which a sender A synchronizes to a receiver B with the help of intermediate forwarding nodes. The intermediate nodes exchange messages but do not synchronize actively. The shortest path between these two nodes requires n hops, through the nodes $\{G_i\}_{i=1}^{n-1}$.

Secure Simple Multi-hop Synchronization (SSM)

1. A$(T1) \rightarrow G_1 \rightarrow \ldots \rightarrow G_{n-1} \rightarrow (T2)$B: $A, B, N_A, sync$
2. B$(T3) \rightarrow G_1 \rightarrow \ldots \rightarrow G_{n-1} \rightarrow (T4)$A:
   $B, A, N_A, T2, T3, ack, MAC_{K_{AB}}[B, A, N_A, T2, T3, ack]$
3. A calculates delay $d = \frac{(T2-T1)+(T4-T3)}{2}$
   If $d \leq d_M^*$ *then* $\delta = \frac{(T2-T1)-(T4-T3)}{2}$ *else* abort

Note that this protocol assumes that there is a secret key K_{AB} between nodes A and B, 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 end-to-end delay by a larger value $d_M^* \gg d^*$. Like in the one-hop time synchronization protocol, the variance of d_M^* will determine how fine-grained the synchronization is. It will also set an upper-bound on the attacker's possible impact on the synchronization precision.

The end-to-end delay in SSM 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 inside the nodes can be neglected, as they are two-three orders of magnitude smaller than transmission and MAC access delays. Although the transmission delays can be accurately predicted from the radio speed, the MAC access times can be very unpredictable, ranging from a few microseconds to a few minutes. Note that in contrast to SPS the forwarding nodes introduce MAC access delays which are unknown to the nodes being synchronized.

We performed an empirical study to find out the delay variations over multiple hops in a typical sensor network. The results were obtained by simulations using the Avrora simulator [Titzer et al. 2005]. We used the same embedded code, network stack, time-stamping, etc., as in the previous section. Table IV

Table IV. Statistics of the End-to-End Delay over Multiple Hops

Hop Distance	Maximum Delay (μs)	Minimum Delay (μs)	Average Delay d_{avg}^{M} (μs)	Standard Deviation σ_M (μs)
2	32,094	18,761	25,120	3,861
3	62,926	37,510	49,940	5,450
4	92,509	56,260	74,781	6,738
5	120,841	76,259	99,667	7,827
6	149,174	97,092	124,393	8,841

summarizes the results for a varying number of hops between the sender and the receiver. The statistics were obtained from 1000 independent runs. It can be observed from Tables I and IV 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 single link. Moreover, these two quantities depend highly on the intensity of the network communication traffic and the underlying channel conditions, thereby making it infeasible to estimate a consistent range for the expected end-to-end delay. Given this flaw, we present a better solution in the next section.

6.2 Secure Transitive Multi-hop Synchronization (STM)

The Secure Transitive Multi-hop (STM) synchronization is essentially performed as SPS pairwise synchronization along all the hops on the path from the source to the destination. Without loss of generality, we present our solution using a representative example: A - G_1 - G_2 - B. In this example, node A is trying to synchronize with node B; all intermediate nodes G_i also get synchronized during the execution of STM. The shortest path between these two nodes is three hops, through nodes G_1 and G_2. In STM, the proper scheduling of node synchronization is achieved 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. Unlike SSM, STM requires only neighboring nodes (A and G_1, G_1 and G_2, G_2, and B) to share pairwise secret keys, which is consistent with our system model. There is no need for A and B to share a secret key.

Secure Transitive Multi-hop Synchronization (STM)

1. A $\rightarrow$ G_1 $\rightarrow$ G_2 $\rightarrow$ B: $A, B, N_A, sync$
2. B : $m1 = (B, G_2, notify)$, $M_1 = MAC_{K_{BG_2}}[B, G_2, N_A, m1]$
   B $\rightarrow$ G_2 : $B, G_2, N_A, m1, M_1$
3. G_2 sync to B (SPS)
   G_2 : $m2 = (B, G_2, G_1, notify)$, $M_2 = MAC_{K_{G_2G_1}}[G_2, G_1, N_A, m2]$
   G_2 $\rightarrow$ G_1 : $G_2, G_1, N_A, m2, M_2$
4. G_1 sync to G_2 (SPS)
   G_1 : $m3 = (B, G_2, G_1, A, notify)$, $M_3 = MAC_{K_{G_1A}}[G_1, A, N_A, m3]$
   G_1 $\rightarrow$ A: $G_1, A, N_A, m3, M_3$
5. A sync to G_1 (SPS)

The resiliency of STM to external attackers is comparable to the one for SPS. The SPS protocol is run on each link independently and, hence, the threshold

verification is divided into stages. Every link is evaluated separately using the same maximal delay d^*, as in SPS. Therefore, unlike SSM, STM does not require the estimation of any additional delay parameter. We note that this local verification gives some extra leverage to the attacker. She can introduce multiple pulse-delay attacks on every link simultaneously; the cumulative sum of these attacks can be huge. Note that the pulse delay introduced at every link is at most 12σ as we run SPS on every link. Therefore, the maximum impact of an external attacker for an N-hop network is $6\sigma N$. Similarly, the synchronization precision is given by $\epsilon \sim N(0, N\sigma/\sqrt{2})$. Our earlier work [Ganeriwal et al. 2005] provides a detailed comparison between SSM and STM and proposes another protocol, Secure Direct Multihop synchronization (SDM). SDM is based on the primitive of diffusion, requires fewer packet transmissions than STM and achieves the same synchronization precision in a non-malicious setting, but is not resilient to external attackers as STM.

6.3 Resiliency to Internal Attackers

STM relies on the assumption that all intermediate nodes are trustworthy. Even a single compromised node can bring detrimental effects to the functionality of the complete protocol. The precision of the multi-hop protocols relies on the security of the used topology discovery or routing protocol. Attacks on routing protocols in sensor networks and countermeasures are described in Karlof and Wagner [2003]. Inherent to our protocols is that attacks compromising one or several nodes may increase the shortest path length and, thus, give the attacker time to tamper with the time synchronization. However, the attacker cannot influence the maximal accepted delay ($6\sigma N$ for STM), since it is a network- and device-dependent variable, calculated for an undisturbed network setting. Hence, the attacker cannot falsify the precision of the time synchronization by more than the maximal delay.

A possible protection against compromised nodes can be achieved by introducing redundancy. The two nodes calculate multiple values for the clock offset across mutually disjoint paths. The sender node can then use outlier detection mechanisms to discard the inconsistent values of the calculated clock offset. Clearly, this relies on the assumption that only a minority of the paths will be compromised. This solution is analogous to secure route discovery in ad-hoc networks in the presence of compromised or faulty nodes within the network [Yaar et al. 2003; Hu et al. 2002; Papadimitratos and Haas 2002]. Manzo et al. [2005] provides a detailed study of the impact of attacks from compromised nodes on the multi-hop time synchronization accuracy in sensor networks. The authors propose two candidate outlier detection schemes: RANSAC and LMS. We argue that STM used in conjunction with RANSAC or LMS provides an efficient solution for secure multi-hop synchronization in sensor networks. STM would be run on every disjoint path to establish clock offset relationships and then either RANSAC or LMS can be used for removing inconsistent clock offsets. Here, STM provides resiliency against external attackers and RANSAC/LMS provide resiliency against internal attackers.

7. DISCUSSION: NETWORK-WIDE SYNCHRONIZATION

Network-wide synchronization can be achieved in two stages: (1) A hierarchical tree is established in the network with a reference node as the root, and (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 subtree 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 the rest of the nodes in the network. Thereby, a similar solution of synchronizing along multiple disjoint paths and then using outlier detection mechanisms such as RANSAC and LMS to remove inconsistencies can be used to provide network-wide synchronization. We note that on a network level this means that the reference node needs to maintain multiple trees simultaneously. In Sun et al. [2006a], authors specifically analyze the security flaws of the existing protocols against internal adversaries and provide a solution for secure network-wide synchronization. They provide a detailed simulation study to understand the impact and the resiliency brought into the process of time synchronization by establishing multiple disjoint paths between the reference node and other nodes in the network. We note that using SPS or STM in conjunction with the schemes proposed by Sun et al. can provide a complete solution for secure network synchronization in sensor networks.

An alternative approach for network-wide synchronization based on SPS is the additional application of secure group synchronization (SGS). In order to detect misled branches originating from compromised motes, the nodes can run regular group synchronizations with the nodes of neighboring branches. Group synchronizations are a dynamic and local means to detect incorrect synchronizations in a branch. However, nodes (of correctly synchronized branches) are at risk of being temporarily desynchronized (by nodes of faulty branches) until the branch gets corrected. Countermeasures and remedies can be provided by primitives such as multiple base stations (representing multiple, mutually synchronized reference nodes), multipath routing protocols, or randomized multi-hop synchronizations with distant notes in the network. Using hierarchy-dependent STM together with hierarchy-independent SGS combines two approaches that are denoted as level-based clock synchronization and diffusion-based clock synchronization in Sun et al. [2006a].

8. CONCLUSIONS

Existing solutions for time synchronization in sensor networks are not resilient to malicious behavior from external attackers or to internally compromised nodes: we showed the feasibility of a pulse-delay attack, where an attacker

Table V. Summary of the Secure Time Synchronization Protocols (on Mica2 Motes). σ is the Standard Deviation of the End-to-End Delay

	Pair of Nodes		**Group Sync.** (N nodes)		**Multi-hop**
Protocol	**SPS**	**E-SPS**	**L-SGS**	**SGS**	**STM**
Number messages	2	3	$2N$	$3N$	$4N$
Resiliency to internal attackers	-	-	No	Yes	No
Sync. Precision	3σ ($10\mu s$)	3σ	3σ ($10\mu s$)	$(6\lfloor\frac{N-1}{3}\rfloor+4)3\sigma$	$3\sigma N$
Attacker impact	12σ ($40\mu s$)	12σ	12σ ($40\mu s$)	$(6\lfloor\frac{N-1}{3}\rfloor+4)12\sigma$	$6\sigma N$

can introduce arbitrarily long delays in the packet propagation time, directly affecting the achieved synchronization precision. We then proposed a suite of protocols for secure pairwise (SPS) and secure group synchronization (SGS) of nodes. We provided an in-depth analysis of these protocols, both in nonmalicious and malicious settings, and implemented them on Mica2 motes. We also carried out an extensive experimental study to gauge the synchronization error performance of these protocols in a malicious setting by emulating external and internal attackers.

SPS achieves the same synchronization precision on a pair of motes as the insecure time synchronization protocols. Even under a pulse-delay attack, SPS can keep the nodes in sync within $40\mu s$. Furthermore, the attacker should be equipped with sophisticated hardware to carry out a valid pulse-delay attack against SPS. We showed that this is not feasible for a mote-class attacker.

SGS uses the SOM algorithm, motivated by Byzantine agreement protocols, to achieve synchronization among a group of nodes even in the presence of internal attackers. We were able to synchronize a group of four motes within $50\mu s$, even when one of them was emulated as an internal attacker. Furthermore, we showed the feasibility of using SGS for larger groups; the SGS algorithm can complete within 10 seconds for groups of 10 nodes. The algorithm is theoretically guaranteed to converge if less than one third of the nodes in the group are faulty or compromised.

SPS can be extended to synchronize nodes that are multiple hops away from each other (STM). The corruption of pairwise relationships between two distant nodes by an internal attacker en-route is protected by establishing multiple relationships along disjoint paths; inconsistent readings are then discarded using outlier detection mechanisms. To conclude, SPS, SGS, and STM provide a secure time synchronization toolbox for sensor networks; their properties being summarized in Table V.

ACKNOWLEDGMENT

The work presented in this article was supported in part by the Swiss National Science Foundation under Grant 200021-116444 as well as by the ONR under grant N00014-06-1-0253, by the U.S. ARL and the U.K. MOD under Agreement Number W911NF-06-3-0001, by the NSF under award CNS-0520006, and by the Center for Embedded Networked Sensing at UCLA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily react the views of the listed funding

REFERENCES

BERMAN, P. AND GARAY, J. A. 1993. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Math. Syst. Theory 26,* 1, 3–19.

BRANDS, S. AND CHAUM, D. 1994. Distance-bounding protocols. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology (EUROCRYPT'93)*. Springer-Verlag New York, Inc., Secaucus, NJ. 344–359.

CHAN, H., PERRIG, A., AND SONG, D. 2003. Random Key Predistribution Schemes for Sensor Networks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy (SP'03)*. IEEE Computer Society, 197.

DEB, B., BHATNAGAR, S., AND NATH, B. 2002. A topology discovery algorithm for sensor networks with applications to network management. In *Proceedings of the IEEE CAS Workshop (CCAS'02)*.

DENG, J., HAN, R., AND MISHRA, S. 2003. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. In *Proceedings of the 2nd IEEE International Workshop on Information Processing in Sensor Networks (IPSN'03)*.

DOLEV, D., HALPERN, J., AND STRONG, H. 1986. On the possibility and impossibility of achieving clock synchronization. In *J. Comput. Syst. Sci. 32*, 230–250.

DOLEV, D. AND YAO, A. 1981. On the security of public key protocols. Tech. rep., Stanford, CA.

ELSON, J., GIROD, L., AND ESTRIN, D. 2002. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operat. Syst. Rev. 36,* SI, 147–163.

ERGEN, S. C. AND VARAIYA, P. 2006. Pedamacs: Power efficient and delay aware medium access protocol for sensor networks. *IEEE Trans. Mobile Comput. 5*, 7, 920–930.

ESCHENAUER, L. AND GLIGOR, V. D. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*. ACM Press. 41–47.

GANERIWAL, S., CAPKUN, S., HAN, C., AND SRIVASTAVA, M. 2005. Secure time synchronization service for sensor networks. In *Proceedings of the ACM Workshop on Wireless Security (WSNA'05)*. ACM Press.

GANERIWAL, S., GANESAN, D., SIM, H., TSIATSIS, V., AND SRIVASTAVA, M. B. 2005. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys'05)*. ACM Press.

GANERIWAL, S., KUMAR, R., AND SRIVASTAVA, M. B. 2003. Timing-sync protocol for sensor networks. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys'03)*. ACM Press, 138–149.

GARAY, J. A. AND MOSES. 1998. Fully polynomial byzantine agreement for n > 3t processors in t+1 rounds. *SICOMP: SIAM J. Comput. 27*.

GREUNEN, J. V. AND RABAEY, J. 2003. Lightweight time synchronization for sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications (WSNA'03)*. ACM Press, New York, NY, 11–19.

HALPERN, J., SIMONS, B., STRONG, H., AND DOLEV, D. 1984. Fault-tolerant clock synchronization. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*. 89–102.

HAN, S., KUMAR, R., SHEA, R., KOHLER, E., AND SRIVASTAVA, M. B. 2005. A dynamic operating system for sensor nodes. In *Proceedings of the ACM Conference on Mobile Systems, Applications and Services (MobiSys'05)*. ACM Press.

HOHLT, B., DOHERTY, L., AND BREWER, E. 2004. Flexible power scheduling for sensor networks. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*.

HU, Y.-C., PERRIG, A., AND JOHNSON, D. B. 2002. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'02)*. ACM Press, 12–23.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. ACM Press, New York. 162–175.

KARLOF, C. AND WAGNER, D. 2003. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*. 113–127.

LAMPORT, L. AND MELLIAR-SMITH, P. 1985. Synchronizing clocks in the presence of faults. In *JACM, 32*, 52–78.

LAMPORT, L., PEASE, M., AND SHOSTAK, R. 1982. The byzantine generals problem. In *ACM Trans. Program. Lang. Syst. 4*, 382–401.

LIU, D. AND NING, P. 2003. Location-based pairwise key establishments for relatively static sensor networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03)*.

LUNDELIUS, J. AND LYNCH, N. 1984. An upper and lower bound for clock synchronization. In *Inform. Contr. 62*, 190–204.

MAHANEY, S. AND SCHNEIDER, F. 1985. Inexact agreement: Accuracy, precision and graceful degradation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'85)*.

MANZO, M., ROOSTA, T., AND SASTRY, S. 2005. Time synchronization attacks in sensor networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'05)*. ACM Press.

MAROTI, M., KUSY, B., SIMON, G., AND LEDECZI, A. 2004. The flooding time synchronization protocol. In *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys'04)*. ACM Press, New York. 39–49.

MAROTI, M., SIMON, G., LEDECZI, A., AND SZTIPANOVTIS, J. 2004. Shooter localization in urban terrian. In *Comput. 37*, 60–61.

PAPADIMITRATOS, P. AND HAAS, Z. 2002. Secure Routing for Mobile Ad Hoc Networks. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'02)*.

PERRIG, A., CANETTI, R., TYGAR, J. D., AND SONG, D. 2002. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes 5,* Summer.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLAR, D., AND TYGAR, J. D. 2001. SPINS: Security protocols for sensor networks. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (Mobicom'01)*.

RASMUSSEN, K. B., CAPKUN, S., AND CAGALJ, M. 2007. SecNav: Secure broadcast localization and time synchronization in wireless networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'07)*. New York, 310–313.

SCHALLER, P., ČAPKUN, S., AND BASIN, D. 2007. Bap: Broadcast authentication using cryptographic puzzles. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'07)*. Vol. 4521. Springer, 401–419.

SICHITIU, M. AND VEERARITTIPHAN, C. 2003. Simple, accurate time synchronization for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference*.

SRIKANT, T. K. AND TOUEG, S. 1987. Optimal clock synchronization. In *JACM*.

SUN, K., NING, P., AND WANG, C. 2005. Fault-tolerant cluster-wise clock synchronization for wireless sensor networks. *IEEE Trans. Depend. Secure Comput. 2*, 3, 177–189.

SUN, K., NING, P., AND WANG, C. 2006a. Secure and resilient clock synchronization in wireless sensor networks. In *IEEE J. Selected Area Comm. 24*.

SUN, K., NING, P., AND WANG, C. 2006b. Tinysersync: secure and resilient time synchronization in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06)*. ACM Press, New York. 264–277.

SUNDARARAMAN, B., BUY, U., AND A, K. 2005. Clock Synchronization for Wireless Sensor Networks: A Survey. Tech. rep. March.

TITZER, B., LEE, D., AND PALSBERG, J. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of IPSN Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems (SPOTS'05)*.

XU, W., TRAPPE, W., ZHANG, Y., AND WOOD, T. 2005. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*. Illinois.

YAAR, A., PERRIG, A., AND SONG, D. 2003. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of IEEE Symposium on Security and Privacy (SP'03)*.

Received June 2006; revised August 2007; accepted May 2008