Time Synchronization Attacks in Sensor Networks

Tanya Roosta[1], Mike Manzo[2], and Shankar Sastry[3]

[1] University of California at Berkeley roosta@eecs.berkeley.edu
[2] University of California at Berkeley mike@manzo.org
[3] University of California at Berkeley sastry@eecs.berkeley.edu

In this chapter, we review time synchronization attacks in wireless sensor networks. We will first consider three of the main time synchronization protocols in sensor network in sections. In section we discuss applications of time synchronization in sensor networks. In section we analyze possible security attacks on the existing time synchronization protocols. In section we examine how different sensor network applications are affected by time synchronization attacks. Finally in section we propose possible countermeasures to secure the time synchronization protocols.

1 Introduction

Ad hoc networks are infrastructure-less, possibly multi-hop wireless networks where every node can be either a host or a router, forwarding packets to other nodes in the network. Some applications of sensor networks are in providing health care for elderly, surveillance, emergency disaster relief, and battlefield intelligence gathering. A sensor network consists of anything from a handful to very many tiny wireless devices with sensors. One very popular type of nodes are the motes developed primarily at U.C. Berkeley and Intel, Figure 1. Motes have very constrained resources. An example of a sensor mote is the mica2dot. A typical configuration may have a 4MHz, 8-bit processor, with 128KB of instruction memory, 4KB of RAM, and 512KB of external flash memory. The radio runs at 433 MHz and 38.4 Kbps. Given the limited resources of these sensor nodes, it is a key technical challenge to design secure services, such as time-synchronization.

1.1 Time Synchronization in Sensor Networks

Time synchronization protocols provide a mechanism for synchronizing the local clocks of the nodes in a sensor network. There are several time synchronization protocols for the Internet, such as Network Time Protocol (NTP). However, given the non-determinism in transmissions in sensor networks, NTP cannot be directly used in wireless sensor networks.

Fig. 1. Mica mote family [14]

Time synchronization implementations have been developed specifically for sensor networks. Three of the most prominent are Reference Broadcast Synchronization (RBS) [3] Timing-sync Protocol for Sensor Networks (TPSN) [5], and Flooding Time Synchronization Protocol (FTSP) [10]. However, none of these protocols were designed with security as one the goals. Security is an important issue in sensor networks given their diverse and usually very sensitive applications. For example, it is crucial to protect people's privacy when sensors are used for elderly health care monitoring. Sensor networks are usually unattended after deployment, and their deployment location is un-trusted. In addition, nodes communicate using a radio channel, which makes all communications susceptible to eavesdropping. Therefore, sensor-network security can easily be breached either by passive attack, such as eavesdropping, or active attacks, such as denial of service attacks, which can be launched at, for example, the routing or the physical layer.

To provide more secure wireless communications in sensor networks, Karlof et. al. proposed and implemented TinySec [7], which uses symmetric private key encryption to authenticate and encrypt messages. If an adversary physically captures a node, however, he will gain access to the network-wide key and can participate in the authenticated communication without being recognized as an attacker. In this work we focus on attacks of this type, where the adversary compromises a node and injects erroneous time synchronization information in the network. To the best of our knowledge, there has been no previous published work on time synchronization attacks in sensor networks, their effect on different sensor network applications, or secure time synchronization protocols.

2 System Model

In this section, we define the problem of secure time synchronization and discuss the clock model, communication model, trust assumptions, threat and attacker models, and security goals we wish to accomplish.

2.1 Clock Model

Every sensor node has a notion of time that is based on the oscillation of a crystal quartz. The sensor clock has a counter that is incremented at rate f where f is the frequency of the oscillation. The counter counts time steps, and the length of these time steps is prefixed. The clock estimates the real time $T(t)$, where,

$$T(t) = k \int_{t_0}^{t} \omega(\tau) d\tau + T(t_0) \tag{1}$$

$\omega(\tau)$ is the frequency of the crystal oscillation and k is a constant [17]. Ideally this frequency should be 1, i.e. $dC/dt = 1$. However, in reality the frequency of a clock fluctuates over time due to changes in temperature, pressure, and voltage. This will result in a frequency different than 1. This difference is termed *clock drift*. There are a number of ways to model the clock drift. In addition to frequency fluctuation in one clock, the crystals of different clocks oscillate at different rates. This difference causes what is called the *offset* between two clocks [17].

2.2 Communication Model

In order to perform time synchronization, the network relies on transmitting the time synchronization messages. The messages are sent through wireless channel. We assume that the wireless links are symmetric, meaning if node A hears node B, then node B also can communicate with node A. However, it is worth mentioning that in reality the wireless links are not always symmetric; in fact, it has been shown through experiments that the wireless channel is asymmetric.

2.3 Adversary Model

As stated above, wireless sensor networks use a wireless channel for communications. This gives an adversary a large window of opportunity, from passive eavesdropping to more serious attacks such as message injection.
We mentioned that in sensor networks there are one or more base stations, which are sinks and aggregation points for the information gathered by the nodes. Since base stations are often connected to a larger and less resource-constrained network, we assume a base station is trustworthy as long as it is available. Beside the base stations, we do not place any trust requirements on the sensor nodes because they are vulnerable to physical capture.
While it is possible that the adversary has access to sensor nodes very identical to the ones deployed or has more powerful nodes such as laptops, in this paper we only consider a mote-class adversary. We assume that the nodes are not tamper resistant.
An outsider attacker has no special access to the sensor network, such as passive eavesdropping, but an insider attacker has access to the encryption keys or other code used by the network. We consider only an insider attacker. We assume that the adversary has been able to capture and corrupt a fraction of the total nodes in the

network. The adversary therefore also has access to the secret keys for authorized communication with other nodes.
The goal of the adversary in our setting is to inject false time synchronization information in the network, without being detected by the honest nodes.

3 Time Synchronization Protocols for Sensor Networks

As mentioned in section 1.1, sensor network is used for monitoring different real world phenomena. Since the existing time synchronization protocol do not fit the special needs of sensor network, a number of clock synchronization protocols have been developed to meet the memory and energy constraint of these networks. There are three main ways to synchronize nodes together. In the first approach an intermediate node is used to synchronize the clocks of two nodes together, such as Reference Broadcast Synchronization [3]. The second approach assumes that the clock drift and offsets are linear, and nodes perform pair-wise synchronization to find their respective drift and offset, such as TPSN [5]. In the third approach, one node declares itself the leader, and all the other nodes in the network synchronize their clocks to the leader, such as Flooding Time Synchronization Protocol [9]. In the following sections, we review these three approaches to the problem of time synchronization in sensor networks.

3.1 Reference Broadcast Synchronization

In RBS a reference message is broadcast to two receivers and the receivers synchronize their respective local clocks to each other, Figure 2. Each receiver records its local time when it gets the reference message. Then the two receivers exchange their local times. It is possible to increase the precision of the estimated time by broadcasting m reference messages instead of one, as follows [3]:

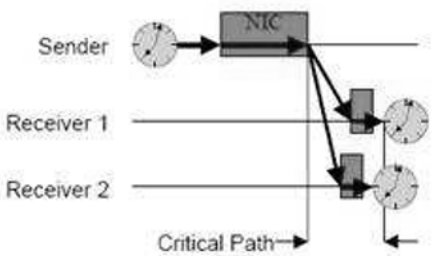

Fig. 2. Example of RBS [3]. RBS does not synchronize the receivers to the sender. Instead it synchronizes the receivers to each other

- A transmitter broadcasts m reference messages.
- Each of the n receivers record the time they received the reference messages based on their respective local times.
- The receivers exchange their local times.
- Each receiver can calculate its phase offset as the LS linear regression of the phase offsets implied by each reference message observed by the receivers as follows: Each node finds the time difference between itself and a neighbor for each reference broadcast message. Then the node finds the least-squares (LS) error fit for these points. The node can then find the offset and skew of its clock from the slope and intercepts point of the line.

The advantage of RBS is that it eliminates the non-determinism on the transmitter side. RBS does not use a multi-hop scheme to synchronize all the nodes to the same global clock. For the multihop case, the nodes that fall within the overlapping region of two reference transmitters perform clock conversion, to go from one clock estimate in one region to the other region. By doing the clock conversion, it is possible to find the time difference between events that happen at different parts of the network.

3.2 Time-Sync Protocol for Sensor Networks

TPSN initially creates a spanning tree of the sensor network. The tree starts at the root of the network, which is generally a base station, and each node establishes its level based on the 'level-discovery' message that it receives. While the tree is being built, the nodes perform pairwise synchronization along the edges of the tree. Each node exchanges synchronization messages with its parent in the spanning tree. By comparing the reception time of the packets with the time of transmission, which is placed on the packet by the parent, the node can find and correct for its own phase offset. We give an example to clarify this scheme. Let us assume there are two nodes, N1 and N2, where N1 is the parent of N2, Figure 3. When N2 wants to synchronize its clock, it sends a *synchronization pulse* to N1 along with the value T1, the time the packet is transmitted from N2.

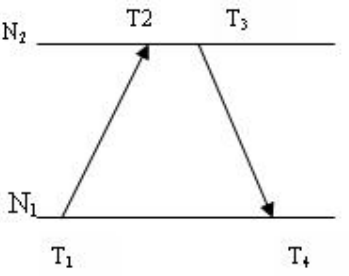

Fig. 3. An example of TPSN

When N1 receives this packet, it sends back an *acknowledgment* packet with times T2, reception time, and T3, retransmission time. N2 receives this ACK packet at T4, and using the four time values it can find the clocks drift and propagation delay, using the equations [5]:

$$d = ((T_2 - T_1) + (T_4 - T_3))/2 \tag{2}$$

TPSN is a sender-initiated time-synchronization protocol, and the sender synchronizes to the receiver's clock. TPSN has a better performance than RBS by time stamping the messages at the MAC layer of the radio stack. It also uses a two way message exchange instead of a one way exchange as in RBS [5]. However, TPSN does not account for clock drift. It also does not efficiently handle dynamic topology changes because it has to compute the spanning tree of the network every time a change happens.

3.3 Flooding Time Synchronization Protocol

In FTSP, a root node broadcasts its local time and any nodes that receive that time synchronize their clocks to that time. The broadcasted synchronization messages consist of three relevant fields: *rootID*, *seqNum*, and *sendingTime* (the global time of the sender at the transmission time). Upon receiving a message, a node calculates the offset of its global time from the global time of the sender embedded in the message [9]. The receiving node calculates its clock skew using linear regression on a set of these offsets versus the time of reception of the messages.
Given the limited computational and memory resources of a sensor node, it can only keep a small number of reference points. Therefore, the linear regression is performed only on a small subset of the received nodes. Since this regression requires that set of updates, however, a node cannot calculate its clock skew until it receives a full of reference messages. Therefore, there is a non-negligible initiation period for the network.
FTSP also provides multi-hop time synchronization in the following manner: Whenever a node receives a message from the root node, it updates its global time. In addition, it broadcasts its own global time to its neighbors. All nodes act in a similar manner, receiving updates and broadcasting their own global time to their neighbors. To avoid using redundant messages in the linear regression described above, each node retains the highest sequence number it has received and the *rootID* of the last received message used. A synchronization message is only used in the regression if the *seqNum* field of the message (the sequence number of the flood associated with that message) is greater than the highest sequence number received thus far and the *rootID* of the new message (the origin of the flood associated with that message) is less than or equal to the last received *rootID*. FTSP is more robust against node failures and topology changes than TPSN since no topology is maintained and the algorithm can adapt to the failure of a root node. If a node does not hear a time synchronization message for a ROOT_TIMEOUT period, it declares itself to be the new root. To make sure there is only one root in the network, if a root hears a time

synchronization message from another root with lower ID than itself, it gives up its root status.

4 Attacks on Time Synchronization Protocols

In this section, we discuss different attacks on the time-synchronization protocols explained above. We discuss possible attacks on RBS, TSPN, and FTSP in each subsection.
All the attacks on time synchronization protocols have one main goal, to somehow convince some nodes that their neighbors' clocks are at a different time than they actually are. Since global time synchronization is build upon synchronization at the neighborhood level, this will disrupt the mechanisms by which the protocols above maintain global time in the network or allow events at distant points in the network to be given to be give time-stamps which reflect the actual difference between their times of occurrence.

4.1 Attacks on RBS

As mentioned above, in RBS a reference message is broadcast by the base station. Two nodes that receive this message exchange their local time for clock synchronization. However, if a node is compromised, it can send a falsified synchronization message to its neighbor during this exchange period. The effect is that the honest node will calculate an incorrect phase offset and skew. In the field of Robust Estimation, the *breakdown point* of an estimator is defined as the smallest fraction of contamination in data that can cause the estimator to take on values arbitrarily away from the real value [16]. It is known that the average and LS estimators have a very low breakdown point. For example, the breakdown point for the LS estimator is $1/n$. Therefore, as n gets larger, the breakdown point gets closer to 0, meaning that a very small fraction of contaminated data can cause the estimation to get far from the real value. If the number of nodes is large in a sensor network, a small fraction of the compromised nodes can cause the time synchronization estimates to get far from the true global time.
It has been proposed [16] to use Least Median of Squares (LMS) instead of LS to fit a more robust model to the data sets. LMS has a high breakpoint of 50%, and therefore is resilient to a high fraction of outliers. LMS is explained in more detail in the Appendix.
It is also possible to attack the multi-hop version of RBS. As mentioned before, RBS does not keep a global time; instead, it tries to find the time difference between the occurrences of two events at different parts of the network. Since the nodes on the boundary of the overlapping regions perform clock conversion, a compromised node placed in any of the overlapping regions can affect multiple regions by giving an erroneous value in the clock conversion. Once the adversary sends a miscalculated clock con-version, this error will be propagated as it is sent throughout the network.

4.2 Attacks on TPSN

TPSN creates a spanning tree of the network at the initial phase of level-discovery and then performs pair-wise clock synchronization. As mentioned above, TPSN is a sender-based synchronization protocol, so the adversary can only affect the protocol through manipulating its children in the spanning tree. That is, the adversary will not be able to cause any disruption by initiating a time-synchronization request since only the parent node will reply. However, the compromised node can effect its children by sending incorrect time stamps for the reception time and the transmit time of the time-synchronization request. The adversary can propagate this desynchronization down the spanning tree toward the leaves on its own branch. Therefore, the closer the compromised node is to the root, the more effect it has on time synchronization in the network, and the larger a region it can contaminate. Alternatively, the adversary can place more compromised nodes at different branches of the tree, and affect a wider region of the network.
The compromised node can also lie about its level in the tree, meaning it can claim a lower level than its real level. By doing so, it can convince the other nodes at its level to request time-synchronization updates from the compromised node. In addition, the compromised node can avoid participating in the tree building phase, and by doing so cut off a number of nodes that would have been its children from the root. If those nodes can-not find any other node as their parent, they will not be able to synchronize to the rest of the network. The effects of this attack would be greatest in a sparse network topology.

4.3 Attacks on FTSP

The major innovations of FTSP, in terms of multi-hop synchronization, over RBS are that the root is chosen dynamically and any node may claim to be the root if it has not heard time updates for a preset interval. One possible attack on this protocol is for the compromised node to claim to be the root node with ID 0 and begin at a higher sequence number than the actual root node so all the updates originating at the actual root node will be ignored. This can be easily done since the protocol allows any node to elect itself root to handle the situation where nodes have not heard from the root node for a long period. Once a compromised node becomes the root, it can give false updates to its neighbors, which will in turn propagate that false time to their neighbors and so on. Every node that accepts the false updates will calculate a false offset and skew for its clock.

5 Effects of Time Synchronization Attacks on Sensor Network Applications

To motivate our discussion of time-synchronization attacks, we describe here in detail the effects of time synchronization attacks on a set of sensor network applications and services that are dependant on time synchronization.

In many application areas, time synchronization allows engineers to design simpler and more elegant algorithms. If security is a high priority, however, the simplest countermeasure against an attack on time synchronization is to build the algorithm such that it does not rely on a time-synchronization service whenever possible.
That said, for certain classes of applications under certain conditions, algorithms cannot provide correct results without an accurate and reliable time-synchronization service. Rather than try to describe those conditions, we have tried to select a representative set of applications that rely on a time synchronization service and demonstrate the effects on them of a time synchronization attack. While we believe these algorithms to be representative of the set of algorithms relying on time synchronization, the set is not exhaustive.

5.1 Shooter Localization

The shooter localization algorithm designed by Vanderbilt University [9,12] has three key stages: Detection: To find the location of the sound of the muzzle blast, the nodes report the time of arrival (TOA) of the sound on three audio frequencies. The sound is processed to determine whether it will be reported to the base station. The report contains an age field. Routing-Integrated Time Synchronization: In this step, the Directed Flooding Framework is used. As the message is routed toward the base station, the routing nodes update the age field by adding the elapsed time between reception and retransmission of the packet. MAC layer time stamping is used, as in FTSP, to update the age field precisely. This age field is used at the base station to find the exact time of the shot [13]. Sensor Fusion: Once the muzzle blast TOA

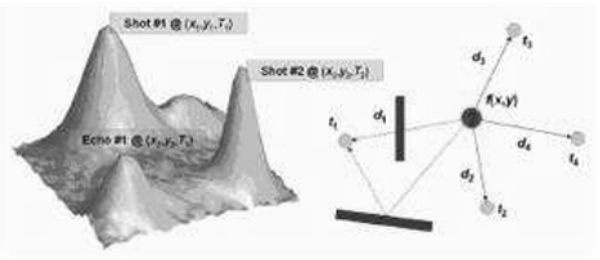

Fig. 4. Consistency function to determine the time and location of the sound from a gunshot [9]

data is submitted to the base station, a sensor fusion algorithm is used to estimate the shooter location and the trajectory of the projectile. The fusion algorithm searches for the maximum of a consistency function, which corresponds to the location of the muzzle blast. The consistency function is defined on four-dimensional space-time space as follows: let (x,y,z) be the shooter position and let t be the shot time. The time of arrival of the ith measurement is [12]:

$$t_i(x, y, z, t) = \frac{\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2}}{\nu} \tag{3}$$

Where v is the speed of sound, and (xi, yi, zi) is the coordinates of the sensor which is making the ith measurement. If the sensor were in a line with the shot, then the real shot time and the estimated time should be equal. However, in reality this does not happen. As a result, the considered measurements are those that satisfy:

$$|t_i(x, y, z, t) - t_i| \leq \tau \tag{4}$$

where $\tau = \frac{\delta_1}{\nu + \tau_2 + \tau_3}$ is the uncertainty value, δ_1 is the maximum localization error, τ_2 is the maximum time synchronization error, and τ_3 is the maximum allowed signal detection uncertainty. It is usually assumed that the uncertainty value is dominated by the localization error and not the other two terms. Therefore, from the above definition the value of the consistency function is defined as the number of observations that support the claim that the shot was taken from location (x, y, z) at time t with uncertainty τ, where it is assumed that an upper bound is known for τ based on the localization, time synchronization and the signal detection algorithms :

$$C_\tau(x, y, z, t) = count_{i=1,\ldots,N}(|t_i(x, y, z, t) - t_i| \leq \tau) \tag{5}$$

The maximum of this consistency function gives the location and time of the shot, as in Figure 4.

However, if the time synchronization protocol has been attacked by compromised nodes, the consistency function can be affected. The maximum time synchronization error might become comparable to the localization error. This might change the shape of the consistency function and, consequently, change the value and location of the local maximum. This would yield an incorrect estimate of the location and time of the shot. For example, if the consistency function is as in Figure 4, there are three local maxima in the consistency function. If, due to a time synchronization attack, some of the nodes that reported time t_1 for the first shot report a shot at $t_1 + \Delta$, the first maximum will be pushed down and the location and time of the shot will incorrectly be estimated at another peak in the consistency function.

5.2 TDMA-based Channel Sharing

In TDMA-based channel sharing protocols, time is divided into intervals and each node is assigned a schedule in which to transmit and receive messages. In this section, we are going to discuss two different protocols that use TDMA-based approaches. The first is the flexible power scheduling [6] and the second is *PEDAMACS* [2].

Flexible Power Scheduling

The goal of this protocol is to reserve time slots for transmission and reception of data messages on each node so that the nodes can go to sleep during the idle slots.

This saves power because nodes do not have to be listening transmissions the entire time. We give an outline of the protocol below:
Time is broken into cycles and each cycle consists of m time slots. During each slot the node has one of the following six states [6]:

- *Transmit (T)*: The node may transmit a message to its parent.
- *Receive (R)*: The node may receive a message from its child.
- *Advertisement (A)*: The parent node broadcasts an advertisement to find an available reservation slot from the child
- *Transmit Pending (TP)*: The node sends a reservation request to its parent.
- *Receive Pending (RP)*: The node receives a reservation request from its child.
- *Idle (I)*: The node powers down during this period if this slot in its schedule is unoccupied.

The algorithm proceeds as follows: When a node is in an A slot, its parent advertises for a reservation slot and marks the corresponding slot in its own schedule as RP. If a child node hears this advertisement, but still has not met its demand, it marks the corresponding slot in its schedule as TP, and sends a reservation request when the time slot arrives. To initialize the algorithm, the base station picks a reservation slot at random and sends out an advertisement for it. It then waits for a reply during the following cycle. When a new node joins the network, it sets all slots in its schedule to I and then listens for one cycle for an advertisement. In order to keep the time synchronized, once a node selects a parent, it periodically synchronizes its time to the parent. This is similar to TPSN in the sense that the child synchronizes its clock to the parent in a spanning tree of the network.
We can summarize the FPS algorithm's main steps during each cycle in the following [6]: After initialization, each node picks an advertisement slot A randomly from the idle slots. Then it picks a reservation slot (RP) at random from the idle slots.
If supply $\geq$ demand

- Turn off the radio during the idle slots
- Schedule an advertisement during an A slot

Otherwise,

- Leave the radio on and listen for advertisement Schedule a reservation request during a TP slot

For each time slot, the node checks its power schedule, and performs an action according to its schedule. For example, if the slot is marked T, it transmits a message. At the end of the cycle, the node clears the current slot and the previous RP from the schedule.
It is worth mentioning that the reservations are not renegotiated unless the child node cancels the reservation or the parent node has a time-out due to having no transmissions for a given period of time. In addition, if the parent node does not receive a message from a child for a large number of cycles where it has R in its schedule, it assumes there has been a change in the topology of the network, and the R slot is

recycled in the parent schedule [6].
If the compromised node is a parent, it can send wrong time synchronization messages, and that will cause the schedule of the child to be in-consistent with the parent's schedule, i.e. the time slots corresponding to the same action in the two schedules will not be aligned in time. Since the parent can send wrong messages to different children, the time slots in multiple nodes will conflict with each other, and as a result, there will be more collisions due to transmission at overlapping time slots. Even though the protocol does not need a fine-grained time synchronization protocol to work, when the error in local clocks gets accumulated over time due to an attack, it can affect and degrade the performance of FPS protocol.

PEDAMACS

PEDAMACS is a TDMA-based MAC protocol used in sensor networks. It assumes that the base station is attached to a reliable and infinite power supply and is able to reach all the nodes in the network. The protocol consists of three phases [2]:

- Topology discovery
- Topology collection
- Scheduling

The network uses three different ranges for communication, where the longest range is used by the base station to reach all the nodes in one hop. The second range is used to determine the interference with other nodes, and the last level is for a node to reach its one hop neighbors. Topology discovery is done using Carrier Sense Multiple Access (CSMA). In the topology collection phase, the base station gathers information about the nodes in the network and builds a tree of the shortest paths. Next, the base station creates a transmission schedule based on its knowledge of the network topology and broadcasts it to all the nodes. This schedule is structured such that nodes which do not interfere with each other during transmission may transmit in the same time slot. The base station periodically broadcasts the current time to all the nodes in the network to facilitate this scheduling. A guard interval on the time slots is used to compensate for clock drift between updates to the current time [2].
A compromised node might spoof the base station and broadcast an alternate time to its neighbors or it might attempt to jam the network to prevent the time updates from being received. If the induced difference in time between 2 nodes that are trying to transmit on adjacent slots is greater than the guard interval, packet loss may occur due to contention in the channel.
Since the base station can, by definition, communicate directly with every other node without routing packets through intermediary nodes, the base station can simple use an authenticated broadcast protocol to prevent a compromised node from injecting false packets. We see no counter-attack to a jamming attack.

Estimation

To illustrate the effects of corrupted time synchronization on estimating state based on sensor readings from a sensor network, we give a simple example using the

Kalman filter. The Kalman filter estimates the state of a discrete-time controlled process that governed by a linear stochastic difference equation [13]:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \qquad x \in \Re^n \tag{6}$$

given the measurement $z_k \in \Re^m$, where

$$z_k = Hx_k + v_k \tag{7}$$

The random variables w and v represent process and measurement noise and are assumed to be independent random variables with normal distribution,

$$p(w) \sim N(0, Q) \qquad p(v) \sim N(0, R)$$

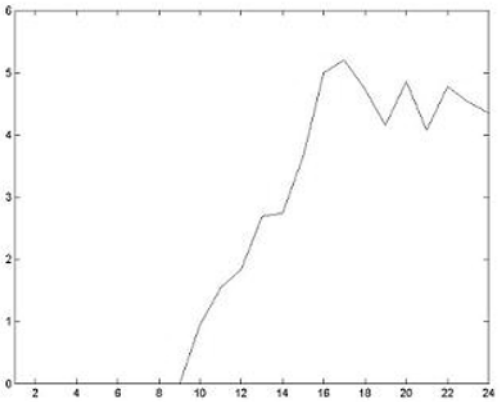

Fig. 5. The y axis shows the norm of the difference between the results from the Kalman filter before and after de-synchronization. The x axis is the time of the corresponding observation.

The Kalman filter estimates the state at every time step. We simulated the movement of an object using 7, where the state is position and velocity of the object in two dimensions. We then used the Kalman filter to estimate the position and velocity of the object before and after modifying the time of some of the position observations, as might occur in an attack on the time synchronization in the sensor network. The norm of the error is shown in Figure 5. As seen in the Figure 5, we began the de-synchronization at time 10.

Authenticated Broadcast(μTesla)

μTESLA is an authenticated broadcast protocol for sensor networks that forms the basis for some of countermeasures proposed in this paper. It relies on an asymmetric

mechanism to allow nodes receiving a broadcasted message to verify the authenticity of the message. That mechanism is based on the arrival time of the messages, so one of the requirements of μTESLA is that the base station and the nodes have loose time synchronization.

Before beginning a broadcast that the nodes in the network may authenticate, the base station must bootstrap the protocol. Each message is transmitted in a time slot, although more than one message may be transmitted in one slot, and each time slot requires exactly one key. So the base station must transmit to each of the other nodes, authenticated with a private key shared only by the base station and that node, the length and starting time of the time slots. It also transmits a private key K_0. To generate K_0, the base station first generates a single key K_n and keeps it private. It then generates enough additional keys from K_n to transmit the packets in a sequence of time slots.

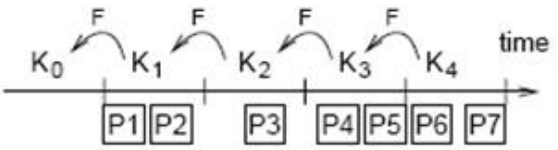

Fig. 6. Example of $\mu TESLA$ [11]

For example, if only one packet can be transmitted for each time slot, the base station would need $|S|$ additional keys. To generate a key from K_n, the base station applies a public one-way function F to K_n once for each of the required keys. That is, $K_i = F(K_i + 1)$. The sequence of keys constitutes a one way key chain. K_0 is last key in this sequence and is used to authenticate the messages in the first time slot, with K_i used for the ith interval and so on [11].

The nodes retain the packets they receive from the base station because they cannot authenticate them immediately. K_i is kept secret until after the *ith* interval. That is, at the $(i + k)th$ interval, for some k transmitted during the bootstrap process, the base station reveals K_i. With that key, the nodes can authenticate all the messages received at interval i (and all the messages in the previous time slots) [11].

To illustrate how the protocol works, consider Figure 6. P_1 through P_7 are the data packets, and K_0 through K_4 are the keys of the one way key chain. K_1 is used to decrypt P_1 and P_2, and K_3 is used to decrypt P_4 and P_5.

It is easy to see that μTESLA is vulnerable to time synchronization attacks. Although the protocol only requires loose clock synchronization, it does require that the network have a synchronized time. That is, the nodes must be able to unambiguously determine in which slot the messages occur. If a compromised node injects

corrupted time updates that exceed the time synchronization error bound, the honest nodes would not be able to decide to which slot the messages or the keys belong. As a consequence, they would be unable to authenticate messages from the base station. It is interesting to note the symbiotic relationship between our countermeasures and μTESLA. On the one hand, our countermeasures rely on μTESLA or some other authenticated broadcast scheme. But without our countermeasures or some other scheme for securing time synchronization protocols, μTESLA is not secure. This is not to say that neither can function since they cannot function until the other has been established. Rather, if our countermeasures maintain time synchronization, they can continue to use μTESLA to do so.

6 Countermeasures for Time Synchronization Attacks

We propose countermeasures for single hop networks and multi-hop networks separately. Each time synchronization protocol facilitates single hop synchronization by having a node periodically broadcast its local version of the global time and allowing the other nodes to synchronize their clocks to that time. Since multihop time synchronization algorithms are all extensions to that basic building block, our proposals for multihop networks are a superset of our proposals for single hop networks.

6.1 Countermeasures for Single Hop Networks

RBS is intended to be used in single-hop networks, although there is a multi-hop extension of the protocol. TPSN and FTSP can also be run in a single-hop network, however, and reportedly perform better than RBS. Regardless of which of these protocols is used in a single-hop network, however, the scheme we present is applicable since in the single-hop scenario all three protocols behave similarly.

In single-hop networks, every node is within radio range of every other node. We consider separately single-hop networks in which all the nodes are within range of a base station, a locus of trust, and those networks in which there is no such base station. The former is the case in many small deployments or deployments that include a second tier of supernodes. Such supernodes may maintain their time using GPS or by communication with another network that has a reliable time synchronization service. For these networks, the central challenge for time synchronization security is to prevent any node from spoofing the root node and sending erroneous updates to the global time.

Such a network might employ a network-wide symmetric private key to encrypt and authenticate messages from the root node, including time synchronization updates, to prevent spoofing of the root node and falsification of the time updates. There exist implementations of such a scheme for sensor networks, as mentioned above. An insider attack might occur, however, if a node were physically compromised. An adversary would gain access to the network-wide key and could falsify time synchronization updates. To detect the presence of falsified time updates, the nodes could easily look for redundant sequence numbers in the packets. Or, if the updates were expected

to be sent at a regular, predefined period, the nodes could detect injected packets by a change greater in the frequency of the time-synchronization updates greater than would reasonably be expected shifting network conditions. If a node detects the presence of falsified packets, it could rely on a private key shared only with the base station to request and authenticate packets from the base station. However, such a scheme would require each node in the network to have a private key with the base station, a highly inefficient proposition. Moreover, the radio traffic would increase linearly with the number of nodes requesting authenticated time updates. For these reasons, we recommend an authenticated broadcast scheme such as μTESLA as an effective means of maintaining time synchronization in single-hop networks with a base station. The base station should encrypt all time synchronization updates.

In single-hop networks that lack a base station or in which the base station fails, there is no trustable source of global time. In that situation, the nodes can elect a root node against which the other nodes can synchronize their clocks. FTSP provides one mechanism for electing a root node, as discussed above. Upon election, the new root node can bootstrap a new key chain for μTESLA using pair wise keys with the other nodes. How-ever, we see no way to prevent a corrupted node from being elected as the root node under the FTSP scheme. Is fact, under the FTSP scheme, a corrupted node could become the root with certainty, as discussed in the attack section. A corrupted root node could send erroneous reference broadcasts that would cause the nodes to calculate an erroneous skew and offset. To prevent this, we propose that, instead of allowing a single node to be the source of time synchronization updates, a subset of the nodes act as the root on a rotating basis. In addition, to prevent a corrupted node from impersonating other nodes, we recommend that all the nodes share a private key with that subset of the nodes in the single-hop network that may become a root. That is, if there are N nodes in the network and M $\ll$ N nodes may become a root node, there will be $N * M$ private keys in the entire network. Those keys could be used to bootstrap an authenticated broadcast scheme for each of the nodes in the rotation of root nodes. Any corrupted nodes might continue to send erroneous updates under this scheme, but the effects on the nodes' calculations of the skew and offset would be reduced. For instance, if there are C corrupted nodes in the network and every node has equal probability of being elected as a root node (a generous assumption, based on the previous discussion of the FTSP root-election scheme), then the probability that a corrupted node is elected a root node is C/N. Suppose the nodes use linear regression with L data points to calculate the clock skew. Then without a root rotation scheme, with probability C/N, all the data points would come from a corrupted node.

If we let M nodes be the root in a round robin fashion, then the probability of having at least one corrupted node being the root is:

$$p = \frac{\binom{C}{1}\binom{N-C}{M-1}}{\binom{N}{M}} \tag{8}$$

If we use LS to find the slope and intersection of the regression line using k data points, we have:

$$b = \frac{\sum_k (x_i - \overline{x})(y_i - \overline{y})}{\sum_k (x_i - \overline{x})^2} \tag{9}$$

$$a = \overline{y} - b\overline{x}$$

where $\overline{x}, \overline{y}$ are averages. Now if one data point is corrupted, i.e. the value of one of the x_i (and as a result y_i) is shifted by Δ. That will cause the averages to shift by Δ/k. We call this shift in slope and y-axis intercept Δb and Δa. Therefore, if we have one compromised node introducing corrupted data in the network, we have a change of Δb in the clock skew. A compromised node is selected with probability p, so the expected value of the change in clock skew is:

$$E(b) = b * \Delta b$$

6.2 Countermeasures for Multi-Hop Networks

In multi-hop networks, at least one pair of nodes communicates with each via a third node that routes messages between them. As in single-hop networks, there may or may not be a base station, a source of trustable time synchronization updates, although most deployments do have such a base station. However, all the nodes in the network may not be in contact with the base station at all times. While there are various multi-hop schemes for time synchronization, as discussed above, they are all essentially schemes for repeating the synchronization scheme for single hop networks, estimation of a node's clock skew and offset by comparison with neighboring nodes, across larger networks. The difficulty in keeping each of the multi-hop schemes secure is in preventing corrupted nodes from increasing the synchronization error between the node with the reference clock and distant nodes. That is, the difficulty is in enabling distant nodes to verify the time synchronization updates they receive despite relying on nodes between the node with the reference clock and themselves to actively maintain-not simply route those updates.

For accurate time synchronization, the nodes must receive the global time of their single-hop neighbors, not nodes that are more distant in the network. If a node computed the offset of its global time from the global time it received from a distant neighbor, that offset would include nondeterministic delays due to the time required to route the update message by intermediate nodes. Therefore, an authenticated broadcast scheme as provided by $\mu TESLA$ does not suffice to securely and accurately synchronize the time of the network to the broadcasting node. The root node could not maintain time synchronization by periodically flooding the network with authenticated clock updates. Clock skew and offset could be calculated in the same way as for updates from a neighboring node. Such updates would, however, provide a useful approximation on the clock skew and offset for all the nodes in the network. Such updates might be sent an order of magnitude less frequently than the updates from a node's immediate neighbors. For each node, the error in both the skew and offset derived from these updates would be primarily determined by the number

of hops between the node and the base station, since there would be nondeterministic delays for each hop. In the absence of a base station, the long term trends in the skew calculated by a node might also serve as a useful approximation. So, our first proposal for multi-hop networks is to use such an approximation to get an upper bound on the error that can be induced by an adversary. If the node receives updates from its neighbors that yields a skew and offset sufficiently far from the approximated skew and offset of the root node or the long-term trends, it could ignore its neighbors and use that approximated skew and offset instead. We are currently conducting tests to show the accuracy of such an approximation experimentally.

As discussed above, FTSP and TPSN rely on updates from a single neighbor node to calculate the offset and skew of its clock. One obvious means of increasing the reliability of these synchronization schemes, then, is to introduce redundancy into the system. This is our second proposal for multi-hop time synchronization protocols. In FTSP, it is especially easy to introduce redundancy. Rather than relying on a single update from a single node for each wave of updates from the nearest root node (i.e. for each seqNum), the nodes should record a subset S of the updates from their neighbors. This would increase the storage space required for the linear regression data points by a factor of S. In the current implementation of FTSP, the regression table holds 8 data points of 8 bytes each, 4 for the offset and 4 for the arrival time of that offset. If S were 5, for instance, this scheme would require accommodating 32 additional data points or $32 * (4 + 4) = 256$ bytes. Even on a mote class node, as described above, this is a reasonable additional memory requirement. Given this additional data, the nodes could take the median of the updates for any sequence number instead of whichever update is received first, which is the current scheme in FTSP. The nodes could initially have a set of private keys for their neighbors to authenticate these updates and prevent a compromised node from inserting false updates and corrupting that median. In addition, if a node A saw a trend in which the updates from a neighboring node B tended to deviate from the other neighbors A by more than some threshold, node A could cease considering node B in its calculation of its clock skew and offset.

In that situation, where a node has become skeptical of the updates it is receiving from its neighbors, it may cease sending updates to its neighbors (FTSP) or children (TPSN). This is our third proposal for increasing the security of multi-hop time synchronization protocols, a policy of containment. Under this policy, corrupted nodes would be unable to de-synchronize nodes beyond their immediate neighborhood. However, this policy requires that nodes have multiple sources of time updates in case their source of updates ceases sending updates. In tree-based schemes, such as TPSN, that means having a mechanism whereby the children of nodes in the neighborhood of a corrupted node can find another parent outside the neighborhood of any corrupted node. If the tree for routing time synchronization updates is distinct from the tree for routing any other rout-ing tree used in the network, the tree already satisfies this requirement. However, maintaining multiple trees has the cost of redundant state at each node. In flooding-based schemes such as FTSP, this requirement is satisfied for free since each node receives updates from multiple neighbors and the loss of updates from a single neighbor can be ignored.

In multi-hop networks, if the base station fails or a node becomes disconnected from it, a new root must be elected against which nodes can synchronize their clocks. The same problems arise as discussed above in single hop networks-there is a risk that the newly elected root node will be a corrupted node. To avoid that situation, we recommend the same root rotation scheme as discussed above for single-hop networks. Our final proposal for improving the security of multi-hop time synchronization is to make the LS linear regression used by each node to calculate the skew of its clock more robust. We propose using an algorithm similar to RANSAC [4], which would fit a model to a set of points that contain outliers in the following steps:

- Randomly select a subset of the data points of size m and build the initial model from these points
- Determine the set of data points that are within ϵ of the model and call this set M. This set defines the inliers of the original data set.
- If $|M|$ is greater than a threshold T, we need to re-estimate the model using all the points in M, and the algorithm terminates.
- If $|M|$ is less than T, select a new subset and repeat from the second step on.
- After N trials the largest M is selected, and the model is re-estimated using all the data points in M.

In order to determine how many sample points m we need in each subset, we can use the following formula, where p is the probability that at least one of the subsets does not contain an outlier (usually taken to be 0.99), ϵ is the acceptable proportion of outliers, and N is the number of sub-sets:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^m)} \tag{10}$$

In our case, the outliers are generated by an adversary (or a node with an especially erratic clock). As discussed in the appendix, an alternative to both RANSAC and LS for linear regression that is more robust to outliers is LMS. However, LMS is a complex method and it may not be possible to implement it on a mote-class node. We have no such concerns about RANSAC, since it involves only a moderate modification of the basic LS scheme.

7 Conclusion

In this paper we showed that designing a secure time synchronization protocol is a crucial task in sensor networks by showing the adverse effects of time synchronization attacks on some important sensor network applications.
We described three major time synchronization protocols for sensor networks, the set of attacks on each protocol, and proposed countermeasures for these attacks. We are currently implementing and testing those countermeasures on sensor network testbeds. Based on our results, we can evaluate effectiveness of our proposals, including a comparison of the performance of the RANSAC algorithm to using LMS regression.

References

1. Balogh, G., Ledeczi, A., Maroti, M., Simon, G. Time of Arrival Data Fusion for Source Localization.
2. Coleri, S. PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks. M.S. Thesis, UC. Berkeley, December 2002.
3. Elson, J., Estrin, D. Fine-Grained Network Time Synchronization using Reference Broadcast. The fifth symposium on Operating Systems Design and Implementation (OSDI), p. 147-163, December 2002.
4. Fischler, M. A., Bolles, R. C.. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.
5. Ganeriwawal, S., Kumar, R., Srivastava, M. Timing-Sync Protocol for Sensor Networks. The first ACM Conference on Embedded Networked Sensor Systems (SenSys), p. 138-149, November 2003.
6. Hohlt, B., Doherty, L., Brewer, E. Flexible Power Scheduling for Sensor Networks. Information Processing in Sensor Networks (IPSN), April 2004, Berkeley, CA.
7. Karlof, C., Sastry, N., Wagner, D. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), pages 162-175, November 2004.
8. Ledeczi A., Volgyesi P., Martoi M., et al. Multiple Simultaneous Acoustic Source Localization in Urban Terrain.
9. Maroti, M., Kusy, B., Simon, G., Ledeczi, A. The Flooding Synchronization Protocol. Proc. Of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), November 2004.
10. Oh, S., Russell, S., Sastry, S. Markov Chain Monte Carlo Data Association for General Multiple-Target Tracking Problems.
11. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J. D. SPINS: Security Protocols for Sensor Networks. Mobile Computing and Networking. Rome, Italy, 2001.
12. Simon, G., Maroti, M., Ledeczi, A.. Sensor Network-Based Countersniper System. Proc. Of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), November 2004.
13. Welch, G., Bishop, G. An Introduction to the Kalman Filter. University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, USA. TR95-041. Available online at: http://www.cs.unc.edu/ welch/publications.html
14. Available on the web: www.xbow.com/Products/ productsdetails.aspx?sid=62
15. Available on the web: http://www.cs.unc.edu/ tracker/media/pdf/ SIGGRAPH2001_CoursePack_08.pdf
16. Available on the web: www.wabash.edu/econexcel/LMSOrigin/LMSIntro.doc
17. Rmer,Kay. Time Synchronization in Ad Hoc Networks. Proceedings of MobiHoc 2001, ACM, October 2001.

8 Appendix

In the LMS method, we still find the residuals, $r_i = Y_i - \hat{Y}_i$, as in the LS method, but instead of minimizing the sum of the squared of the residues we do the following:

$$\min med_i r_i^2$$

It is well known that LMS is more robust in the presence of outliers, so it is especially appropriate for this application of regression where data points from corrupted nodes appear as outliers.
Finding the least median squares is a challenging optimization problem, however, since we have to optimize the following equation [16]:

$$\min_{b_0,b_1} med_i SR_i = median\{(Y_1 - (b_0 + b_1 X_1))^2, ..., (Y_n - (b_0 + b_1 X_n))^2\}$$

There are software packages that can perform this optimization, but it may not be possible to port the algorithms to a mote-class node.