Testbed Implementation of A Secure Flooding Time Synchronization Protocol

Tanya Roosta[1], Wei-Chieh Liao[2], Wei-Chung Teng[2], and Shankar Sastry[1]
Dep. of Electrical Engineering and Computer Science, UC Berkeley [1]
Dep. of Computer Science and Information Engineering, National Taiwan University of Science and Technology[2]
{roosta,sastry}@eecs.berkeley.edu[1], {M9415016,weichung}@mail.ntust.edu.tw[2]

***Abstract*— A fundamental building block in distributed wireless sensor networks is time synchronization. Given resource constrained nature of sensor networks, traditional time synchronization protocols cannot be used in sensor networks. Previous research has focused on developing various energy efficient time synchronization protocols tailored for these networks. However, many of these protocols have not been designed with security in mind. In this paper, we describe FTSP which is one of the major time synchronization protocols for sensor networks. We outline the adverse effects of the time synchronization attacks on some important sensor network applications, and explain the set of possible attacks on FTSP. We then propose a number of countermeasures to mitigate the effect of the security attacks. We implement these attack scenarios on a sensor network testbed. We show the extent each attack is successful in desynchronizing the network. Finally, we propose countermeasures and implement them on our sensor network testbed to validate their usefulness in mitigating security attacks. We show that adding a sequence number filter to the original FTSP helps mitigate the effect of attacks on this protocol.**

I. Introduction

Due to recent technological advances, the manufacturing of small, low cost sensors have become technically and economically feasible. Thousands of these sensors can potentially be networked as a Wireless Sensor Network (WSN) for many applications that require unattended, long term operations. Some current applications of sensor networks are providing health care for the elderly, surveillance, emergency disaster relief, detection and prevention of chemical or biological threats, and gathering battlefield intelligence.

One of the critical challenges to making sensor networks more pervasive and secure is the severe resource constraints on the sensor nodes. An example of a sensor node, sometimes called a mote, is the TelosB mote which has a 8 MHz, 16-bit processor, 10 KB of RAM, and 1 MB of external flash memory. It has a 2.4 to 2.4835 GHz, IEEE 15.4 compliant radio with 250 kbps maximum data rate and runs on a AA battery [1]. The limited energy, bandwidth, and computational resources make it difficult to implement security primitives, such as strong cryptography, and implement secure services, such as secure time synchronization.

This work was supported in part by the Team for Research in Ubiquitous Secure Technology at UC Berkeley (TRUST), the National Science Council (NSC), the Industrial Technology Research Institute (ITRI), and the Taiwan Information Security Center at NCTU.

The challenge of resource constraints in sensor networks is compounded with the challenge of designing distributed protocols that can scale from tens to thousands of motes. Another challenge is that typically sensor networks are physically unattended after deployment. As a result, the nodes are vulnerable to physical capture and compromise. All of these issues combined make it difficult to design energy/memory efficient, scalable distributed protocols, such as time synchronization protocols, that are also secure.

In this paper, we look at the security issues in Flooding Time Synchronization protocol (FTSP) proposed in [10]. FTSP is a relatively simple and easy-to-use time synchronization protocol. However, it has not been designed with security as an objective. In this work we show, through experiments run on a real sensor network testbed, that FTSP can become crippled in the face of attacks.

To motivate the discussion of secure time-synchronization, we describe the effects of time synchronization attacks on a set of fundamental 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. A representative set of applications in sensor networks that rely on a time synchronization service are,

- Shooter localization [9]
- TDMA-based channel sharing, such as, Flexible Power Scheduling [8] and TDMA-based MAC protocol [2]
- State estimation [11]
- Authenticated broadcast (μTesla) [12]

To illustrate the effects of corrupted time synchronization, we explain the estimation application in more detail. Many tracking applications use Kalman filter to estimate the state of a moving object based on sensor readings. The Kalman filter estimates the state of a discrete-time controlled process that is governed by a linear stochastic difference equation:

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

given the measurement $z_k \in \Re^m$, where $z_k = Hx_k + v_k$. The random variables w and v represent process and measurement noise and are assumed to be independent random variables with Normal distribution.

The Kalman filter estimates the state at every time step. We simulated the movement of an object using Equation 1,

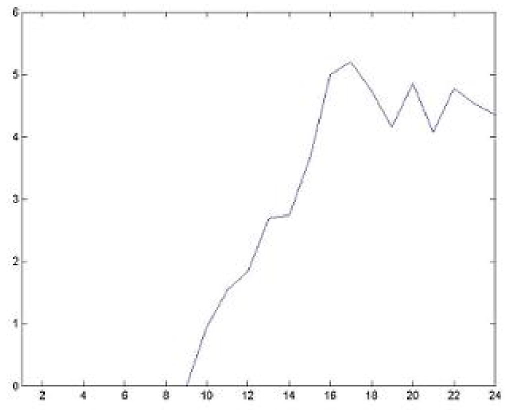

Fig. 1. The y axis shows the norm of the difference between the results from the Kalman filter before and after de-synchronization. The de-synchronization began at time 10. The x axis is the time of the corresponding observation.

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 1.

There have been some work done on secure time synchronization protocols. First is the work by Ganeriwal, et. al [5], which attempts to detect time synchronization attacks. The detection is done using a threshold on the maximum drift and skew of the clock when there are no attacks on the protocol. The algorithm also makes use of the message authentication code (MAC) to ensure the integrity of the time synchronization message updates. In the event of an attack the protocol aborts the time synchronization process. This approach could potentially lead to more problems since an adversary can use this feature to launch a denial of service attack on the sensor network.

The second work is [13] in which the authors employ a combination of pairwise node authentication along with using data redundancy to build a resilient time synchronization protocol. Our countermeasures differ from [5] and [13] in the following important way: we do not abort the time synchronization process when there is an attack, and we do not rely on MAC and cryptographic techniques. Our main objective is to filter out the bad data, coming from the compromised nodes, using more robust methods. We aim to use the existing data more intelligently in order to detect outliers.

The rest of the paper is organized as follows. Section II gives a short background on the concept and necessity of time synchronization in sensor networks and different types of time synchronization protocols in these networks. Section III explains the details of the operations of FTSP protocol. Section IV gives the details of the attack scenarios ran on the sensor network testbed and the result of each experiment on the time synchronization process. Section V describes our proposed countermeasures and the implementation of some of these methods on the testbed followed by the conclusion and future work in Section VI.

II. Background

Sensor networks are to monitor different real world phenomena. Since the existing time synchronization protocols do not fit the special needs of sensor networks, a number of clock synchronization protocols have been developed to meet the memory and energy constraints of these networks. The need for a time synchronization protocol stems from the fact that 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 $T(t) = k\int_{t_0}^{t}\omega(\tau)d\tau + T(t_0)$, where $\omega(\tau)$ is the frequency of the crystal oscillation and k is a constant. Ideally this frequency should be 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*. 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.

There are three general ways to synchronize nodes in a sensor network. In the first approach, an intermediate node is used to synchronize the clocks of two nodes together, such as Reference Broadcast Synchronization (RBS) [3]. The second approach assumes that the clock drift and offsets are linear, and nodes perform pair-wise synchronization, using a tree structure, to find their respective drift and offset, such as TPSN [6]. 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 (FTSP) [10]. Our work focuses on FTSP because of its simplicity and that fact that it has been successfully implemented on a real testbed of sensors.

III. Flooding Time Synchronization Protocol

In FTSP, a root node broadcasts its local time and any nodes that receive the broadcast 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. 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 sensor nodes, they can only keep a small number of reference points (in the current implementation of FTSP 8 data points are saved at each step). Therefore, the linear regression is performed only on a small subset of the received updates. In addition, a node cannot calculate its clock skew until it receives a full set of reference messages (e.g. 8 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 other time synchronization protocols 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 the root gives up its root status if it hears a time synchronization message from another root with lower ID than itself [10].

IV. Attacks on FTSP

In the following subsections, we discuss the attacks scenarios for FTSP that we tried on our sensor network testbed. In all the experiments we conducted, the following attacker model was assumed: the attacker has gained control of a node, possibly through physical compromise, and has the secret keys for participating in legitimate inter-node communication. This is a realistic setting since currently there is no tamper-resistant hardware for the motes [7]. It is possible to add cryptographic protocols to a particular time synchronization protocol to enhance its security features, such as in [13]. However, it is important to realize that once a node is physically compromised, the attacker will gain access of the cryptographic keys and can participate in network communication in a legitimate fashion.

In general, all attacks on any of the existing time synchronization protocols have one main goal, to somehow convince a subset of 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 time stamps that reflect the actual difference between their times of occurrence.

A. Attack on '*sendingTime*'

The first attack scenario in our set of experiments was the attack on the *sendingTime* (see Section III). A subset of nodes were compromised. In this scenario, a compromised node is programmed to send incorrect *sendingTime* to their direct neighbors. The goal was to find out if the compromised nodes are capable of corrupting their neighbors' synchronization process. The result of the experiments shows that this attack is successful if the compromised node can send its synchronization updates sooner than good (non-compromised) nodes in its neighborhood. As a result, if some of the good nodes could send their time updates before the compromised nodes, the corrupted time updates will be discarded due to the *seqNum* feature of FTSP. Figure 2 shows an example of an unsuccessful attack on *sendingTime*.

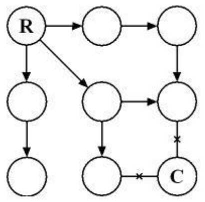

Fig. 2. R is the root node and C is the compromised node. If the neighbors of C receive synchronization message form other good nodes before C, the incorrect *sendingTime* of the compromised node would not have an effect because of the out-of-date *seqNum*.

B. Attack on Time Synchronization Rate

In FTSP, the root node sends a time synchronization update once every TIME_SYNCRATE seconds. As a result, other nodes in the network will receive these update messages at the same rate of 1 message per TIME_SYNCRATE second. A possible attack scenario is that the compromised node would send time synchronization updates more frequently than TIME_SYNCRATE so as to increase the possibility of affecting the time of its neighboring nodes. At the same time, the compromised node must increase the sequence number accordingly to convince its neighbors to consider the compromised node's update message in their regression table (only the highest *seqNum* is considered in FTSP regression table). Otherwise, as Figure 3 shows, the compromised node (B) will not have any affect on the time synchronization of its neighbors. The nodes in the neighborhood of A and B will receive node A's update before B, and since these updates have a higher sequence number, node B's updates are ignored. We implemented this attack on the sensor network testbed, and the result of the attack was in fact successful. Due to the similarity of the plot to Figure 5, and lack of space, we omit the plot of the results of TIME_SYNCRATE attack.

C. Attack on '*seqNum*'

As mentioned in Section III, only the root node is allowed to increase the value of the *seqNum* field of the time synchronization updates. This procedure is to facilitate the dynamic topology change for FTSP. A nice by-product of the *seqNum* is that it could potentially block some of the incorrect time synchronization updates propagated by compromised nodes (Section IV-A). Therefore, the next attack scenario we tried on the testbed was to have the compromised node simultaneously falsify the *sendingTime* and the *seqNum*. This attack scenario

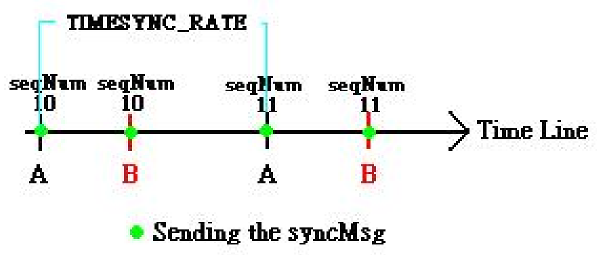

Fig. 3. B is the compromised node and node A is a good node. Both nodes A and B have the same TIME_SYNCRATE, so if node A sends its time updates first, node B's updates will not affect the time synchronization of other nodes due to the *seqNum*.

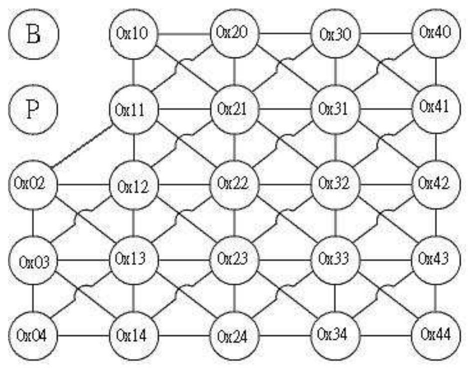

Fig. 4. Testbed used in seqNum attack.

was very successful and resulted in crashing the root node and FTSP completely. The testbed we used, shown in Figure 4, had 25 TelosB motes.

In this attack scenario, we programmed the compromised node to add a 'BAD_SEQNUM' to the *seqNum* of its time synchronization updates. In addition, the compromised node altered the value of *sendingTime* field when sending the time updates to its neighbors. From time $t_0 = 0$ to $t_1 = 180$ second, the motes were going through the process of root selection and initial data gathering (8 data points) for performing the synchronization. At time $t_2 = 600$ seconds, the compromised node was added to the experiment. The effect of this attack can be clearly seen from Figure 5. The attack took effect at $t_3 = 630$ seconds, as shown by the blue points which present the global time estimated by the nodes.

V. COUNTERMEASURES FOR FTSP

In this section, we propose a set of countermeasures for the mentioned time synchronization attacks. It is important to note that the network can 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 [13], as mentioned earlier. This approach, however, will not work if a subset of nodes were physically compromised. An adversary would gain access to the network-wide key and could falsify time synchronization updates. That said, the use of cryptographic schemes to secure the communication among nodes is absolutely a necessary step to take in order to secure the sensor network.

FTSP provides one mechanism for electing a root node (Section III). There is no security restriction in FTSP that would prevent a compromised node from becoming the leader, as we showed through our experiments. In order to fix this problem, we propose using one of the standard distributed coin-flipping algorithms that use cryptographic commitments. For instance, each sensor will pick a random value x_i, broadcasts Commit(x_i), then everyone waits for all broadcasts. Finally everyone opens up their commitment and broadcasts x_i. Now one can use $y = Hash(x_1, .., x_n)$ as a random number. For instance, one can compute [y mod n] and designate that sensor as the leader. This picks a random leader, and as long as least one sensor is honest, then the choice of leader will be uniformly distributed across all sensors. This is in contrast to FTSP's root selection process which chooses the node with the smallest node ID. This countermeasure is not implemented on the sensor network testbed since it is not the focus of our work. However, it shows that there are ways to strengthen the security of the root selection mechanism in FTSP.

Once there is a secure procedure in place for electing the leader node, the next step is to develop a built-in mechanism for FTSP so that the algorithm can correct for erroneous data without solely relying on cryptographic solutions. As discussed above, FTSP relies 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 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 earlier, this is a reasonable additional memory requirement. Given this additional data, the nodes could use these updates for any sequence number instead of whichever update is received first, which is the current scheme in FTSP.

The selected subset S can be a randomly selected subset of neighbor nodes. This is loosely based on the idea of RANSAC [4]. RANSAC relies on random sampling selection to search for the best model to fit a set of points that is known to contain outliers. In other words, RANSAC can be considered to seek the best model that maximizes the number of inlier data.

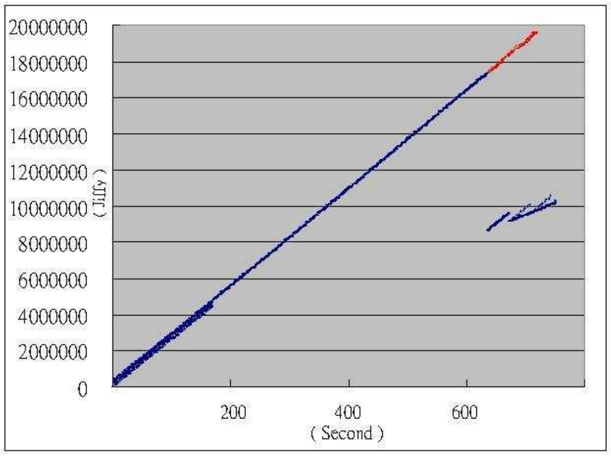

Fig. 5. Result of seqNum attack. The blue line is the result of the actual regression on message updates, and the red line shows what the regression line should have looked like if there was no attack.

A. Countermeasure Implementation: Experimental Results

In this section we describe the results of implementing some of the proposed countermeasures on the sensor network testbed. The first countermeasure implemented on the tested was to filter out bad data by collecting more data point from all the neighbors instead of receiving the synchronization message from only one node, and then filtrating extreme values. To do this, we expanded the data table from 8 entries to 32 entries. The synchronization messages were collected from all neighboring nodes regardless of the message *seqNum*. As a result, the functionality of the *seqNum* field of FTSP was completely ignored.

The result of this attack was surprising and revealed a feature of FTSP which is a side effect of having sequence numbers for each new update message. Although it is claimed in [10] that FTSP does not have any level of hierarchy, in reality it turns out that there, is an implicit hierarchy due to *seqNum*, the local connection and the topology of the network. To explain this point, assume that the root node R broadcasts an update message M_t at some global time t. The nodes which are closest (within the communication range) to R will receive this update first and form the first level of hierarchy. This is due to the fact that in FTSP each node is only allowed to accept the updates with the highest (i.e. most recent) sequence number. We call this set of nodes L_1. These nodes perform the regression step, update their global time accordingly and broadcast their time update messages. The next set of nodes that receives this broadcast is the one-hop neighbors of L_1 nodes, which we call L_2. Continuing in this fashion, it is clear that after i broadcasts of the time update message, there are i sets of nodes (L_i), forming the hierarchy. Nodes in set L_i are one level higher in the hierarchy than nodes in L_{i+1}.

Now when the *seqNum* is not used, this hierarchy breaks down. In addition, the linear relation assumed between the global time and the local time of the nodes, in FTSP, is dependent on the fact that the updates are done in a short period of time where this linearity assumption holds true. However, when we add many data points from multiple neighboring nodes without using the sequence number of each data point in the regression table, the linearity assumption does not hold true anymore. This results in a very unstable linear regression, meaning the result of regression from one time update to the next fluctuates considerably. Figure 7 shows an example where regression is done on three sets of data, and as can be seen, each line has a different slope which results in an unstable regression. If we use all of the data, the regression line will try to fit all of the data, which will not yield a good estimate. This instability results in a significantly less precise time synchronization. Therefore, although our first countermeasure did not aid in preventing attacks, it revealed an important aspect of FTSP not known before, and that is the existence of implicit hierarchy in FTSP and the need for the sequence number as a mean of preserving the linear regression relation and stability of the algorithm.

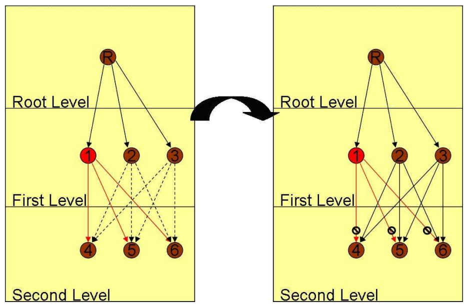

Fig. 6. New data structure could block incorrect seqNum.

Securing seqNum Attack: Given the result of our previous experiment, we need to use a countermeasure that will secure the sequence number functionality. Therefore, we created a new data structure, shown in Figure 6, containing three important parts of the data: the node ID for the incoming synchronization message (IncomingID), *seqNum* corresponding to each IncomingID, and a time out value for the corresponding node ID, called TIME_OUT.

Using this extended data structure, each node randomly chooses k nodes from its total of n neighbors and records these data, instead of recording only one neighbor's data points as in the original FTSP. Then, from among the randomly chosen neighbors, the node chooses the neighbor with the smallest *seqNum* and runs linear regression on the updates received from this node. This is different from the original FTSP where nodes choose the message with the highest sequence number from among 'all' the neighboring nodes. We call this countermeasure *seqNum filter*.

Given we run regression on the set of data points coming from the same node, the linear assumption will hold and the regression is stable. In addition, we keep the *seqNum* to preserve the implicit hierarchy in FTSP. The TIME_OUT feature is used to support the dynamic topology change that FTSP offers. For example, if one of the k neighbors of a node does not broadcast a time update for a period of

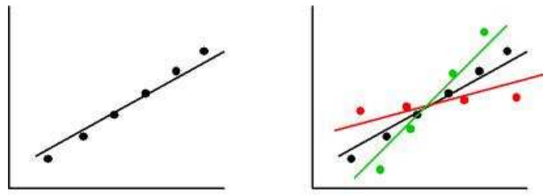

Fig. 7. In the left diagram, the linear relation holds if the synchronization message is received form a single node. In the right diagram, where there more nodes, the regression results lead to inconsistent data from different nodes, and the linear regression would be very unstable.

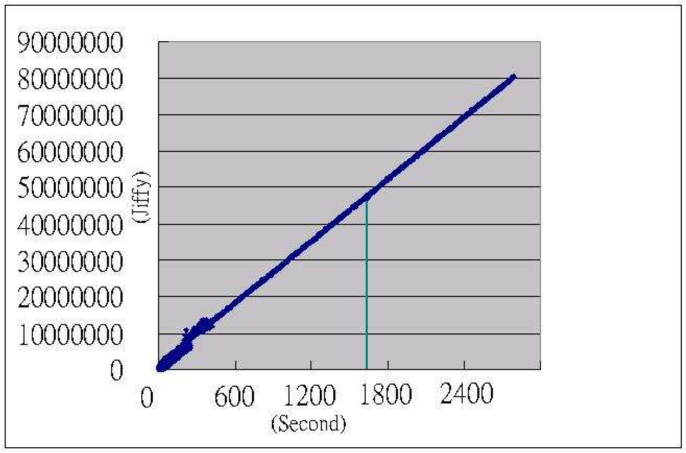

Fig. 8. The seqNum filter used in combination with original FTSP. The attack started at t=1600 second but was unsuccessful.

TIME_OUT seconds, then the node can remove this neighbor from its regression table, and add a new neighbor in its place, which is randomly chosen from among the remaining available neighbors of the nodes. The result of applying the seqNum filter to FTSP is shown in Figure 8. The node ID of the compromised node is 34, shown in Figure 4, and it starts sending wrong time updates at $t = 1600$ seconds. As seen from the plot, the attack clearly did not succeed in falsifying the global time estimation.

The *seqNum* filter proved to be effective in mitigating the effect of attacks on the sequence number and *sendingTime*. Using a combination of random neighbor selection and the seqNum helps make FTSP more robust to insider attacks while preserving the original features of FTSP, such as dynamic topology support.

VI. Conclusion and Future Work

One of the fundamental tasks in sensor networks is the problem of time synchronization. Given the unattended nature of sensor networks, physical capture and compromise of the nodes is possible. Therefore, the attacker gains access to the network cryptographic keys and can participate in the legitimate communication among nodes. Therefore, designing a secure time synchronization protocol is crucial to maintaining the functionality of the sensor networks. In this paper, we described FTSP which is one of the major time synchronization protocols for sensor networks. We outlined the adverse effects of the time synchronization attacks on some important sensor network applications, such as estimation. The set of possible attacks on FTSP protocol was explained next. We then proposed a number of countermeasures to mitigate the effect of the security attacks. The attack scenarios were carried out on a testbed of 25 TelosB motes. We explained the degree to which each attack was successful in desynchronizing the network. Finally, some of our proposed countermeasures were implemented on the testbed to validate their usefulness in mitigating security attacks. We showed that adding the seqNum filter to the original FTSP helps mitigate the effect of insider attacks on this protocol.

References

[1] Crossbow technology: www.xbow.com/products.

[2] S. Coleri. PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. Master's thesis, UC. Berkeley, December 2002.

[3] J. Elson and D. Estrin. Fine-grained network time synchronization using reference broadcast. In *The fifth symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[4] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Communications of the ACM*, volume 24, 1981.

[5] S. Ganeriwal, S. Capkun, C. Han, and M. Srivastava. Secure time synchronization service for sensor networks. In *In Proceedings of ACM Workshop on Wireless Security (WiSe)*, September 2005.

[6] S. Ganeriwawal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *The first ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[7] Carl Hartung, James Balasalle, and Richard Han. Node compromise in sensor networks: The need for secure systems. Technical report, Department of Computer Science University of Colorado at Boulder, 2005.

[8] B. Hohlt, L. Doherty, and E. Brewer. Flexible power scheduling for sensor networks. In *Information Processing in Sensor Networks (IPSN)*, April 2004.

[9] A. Ledeczi, P. Volgyesi, and M. Martoi. Multiple simultaneous acoustic source localization in urban terrain.

[10] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The Flooding Ttime Synchronization Protocol. In *Proc. Of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2004.

[11] Songhwai Oh, Stuart Russell, and Shankar Sastry. Markov chain monte carlo data association for general multiple-target tracking problems. In *In Proc. of the IEEE International Conference on Decision and Control (CDC)*, 2004.

[12] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: Security protocols for sensor networks. In *Mobile Computing and Networking*, 2001.

[13] K. Sun, P. Ning, and C. Wang. Secure and resilient clock synchronization in wireless sensor networks. In *IEEE Journal on Selected Areas in Communications*, volume 24, February 2006.