Securing Flooding Time Synchronization Protocol in Sensor Networks

Tanya Roosta
Dep. of Electrical and
Computer Engineering
UC Berkeley
Berkeley, CA.
Email: roosta@eecs.berkeley.edu

Shankar Sastry
Dep. of Electrical and
Computer Engineering
UC Berkeley
Berkeley, CA.
Email: sastry@eecs.berkeley.edu

***Abstract*—Sensor networks have become popular in the recent years due to their wide range of application. A fundamental building block in distributed wireless sensor networks is Time Synchronization. Because sensor nodes may be severely resource-constrained, traditional time-synchronization protocols cannot be used in sensor networks. Various energy efficient time-synchronization protocols tailored for such networks have been proposed in the recent years. However, none of these protocols have been designed with security in mind. If an adversary were able to compromise a node, he might prevent a network from effectively executing certain applications, such as sensing or tracking an object, or he might even disable the network by disrupting a fundamental service such as a TDMA-based channel-sharing scheme. In this paper we give a detailed explanation of the Flooding Time Synchronization protocol and outline the possible attacks on this protocol. To motivate our work, we briefly discuss how different sensor network applications that are affected by time synchronization attacks. Finally, we propose some statistical countermeasures, as opposed to cryptographic countermeasures, to mitigate the effect of time synchronization attacks.**

I. INTRODUCTION

Ad hoc networks are infrastructure-less, possibly multi-hop wireless networks where every node can act as 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 anywhere 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. Motes have very constrained resources. An example of a sensor mote is the mica2dot (figure 1). 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.

A. 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.
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) [6], and Flooding Time Synchronization Protocol (FTSP) [12]. 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. There are two exceptions to the above non-secure time synchronization protocols: first is the work by Ganeriwal, et. al [7], 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 sync. message updates. In the event of an attack the protocol

Fig. 1. Mica mote family [20]

aborts the time synchronization process. This will lead into more problems since the adversary can use this feature to launch a denial of service attack on the sensor network. The second work is [18] in which the authors employ a combination of pairwise node authentication along with using data redundancy to build a resilient time synchronization protocol. Our work is different from [7] and [18] in the following two ways: we do not abort time synchronization when there is an attack, and we do not heavily rely on MAC. Our main objective is to filter out the bad data, coming from the compromised nodes, using robust regression methods. Second, we do not add redundancy to the data set to make time synchronization more robust. On the contrary, we aim to use the existing data more intelligently in order to detect outliers.

To provide more secure wireless communications in sensor networks, Karlof et. al. proposed and implemented TinySec [9], 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.

The rest of the paper is organized as follows: in section II, we outline the clock, communication, and adversary model used throughout the paper. In section III, we review the Flooding Time Synchronization Protocol (FTSP). In section IV different security attacks on FTSP are explained in detail. The effects of security attacks on a time synchronization protocol are discussed in section V, followed by proposed countermeasures in section VI.

II. 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.

A. 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 [14]. 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 [14].

B. 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.

C. 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.

III. Time Synchronization Protocol for Sensor Networks

As mentioned in section I, 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 (RBS) [4]. 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 [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) [11]. Our work focuses on FTSP in particular since it is a simple time synchronization protocol, and it has been implemented on a real testbed of sensors.

A. *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 [11]. 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 (in the current implementation 8 data points are saved at each step). 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 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.

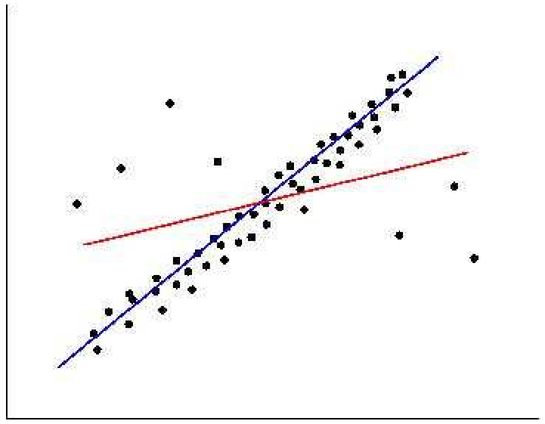

Fig. 2. The effect of outliers on regression. The Blue line is the correct regression line, and the red line is the result of LS regression.

IV. Attacks on the Time Synchronization Protocol

In this section, we discuss different attacks on FTSP explained above. It is worth mentioning that in general, all the attacks on any of the existing 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.

A. *Attacks on FTSP*

The major innovations of FTSP, in terms of multi-hop synchronization, over other time synchronization protocols, is 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.
Another possibility is that the compromised node can forward false time updates to its neighbors. Given each node finds the skew of its clock using Least Squares (LS) method on k data points gathered from its neighbors, even if one data point is corrupted, it will affect the regression line. This effect can be seen in figure 2 where we have outliers in the collected data. When LS is used to find the slope and intersection of the regression line using k data points, we have:

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

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

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. The reason ordinary LS method breaks down in the face of malicious attacks is that the assumptions of independence and normality of the residual errors, which are fundamental to the LS procedure, are violated.

V. 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 the following algorithms to be representative of the set of algorithms relying on time synchronization, the set is not exhaustive.

- Shooter Localization [10]
- TDMA-based Channel Sharing: Flexible Power Scheduling [8],TDMA-based MAC protocol [3]
- Estimation
- Authenticated Broadcast(μTesla) [13]

We explain the third item, estimation, in more detail in the following section, and refer the interested reader to [15] for an extensive discussion on the other items.
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 [19]:

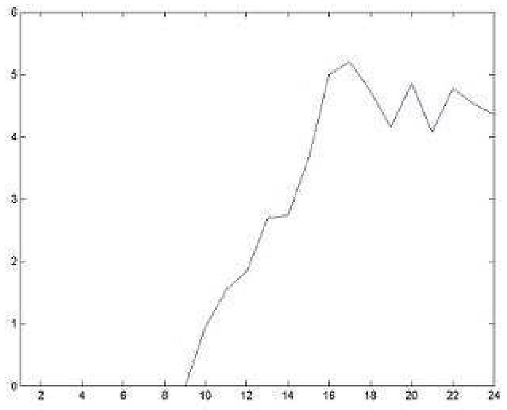

Fig. 3. The y axis shows the norm of the difference between the results from the Kalman filter before and after de-synchronization. The x axis is the time of the corresponding observation.

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \qquad x \in \Re^n \tag{3}$$

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

$$z_k = Hx_k + v_k \tag{4}$$

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

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

The Kalman filter estimates the state at every time step. We simulated the movement of an object using equation 4, 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 3. As seen in the Figure 3, we began the de-synchronization at time 10.

VI. Countermeasures for Time Synchronization Attacks

In this section, we propose a number of countermeasures for the mentioned time synchronization attacks. It is worth noting 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, as mentioned above. This approach, however, will not solve the problem of an insider attack, i.e. if a subset of nodes were physically compromised. An adversary would gain access to the network-wide key and could falsify time synchronization updates.
FTSP provides one mechanism for electing a root node, as

discussed in III. There is no security restriction in FTSP that would prevent a compromised node from becoming the leader. 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 $ymodn$ and designate that sensor as the leader. This picks a random leader, and so long as least one sensor is honest, then the choice of leader will be uniformly distributed across all sensors.

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, 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 third proposal for improving the security of FTSP 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 [5]. 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 effect, RANSAC can be considered to seek the best model that maximizes the number of inlier data. The following is the set of steps taken by the RANSAC algorithm to find the best model parameters:

1) Randomly select a subset of the data points of size m and build the initial model from these points
2) 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.
3) 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.
4) If $|M|$ is less than T, select a new subset and repeat 2.
5) 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 subsets:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^m)} \tag{5}$$

In our case, the outliers are generated by an adversary (or a node with an especially erratic clock).

Our last proposal is to use Least Median of Squares (LMS) as proposed in [16]. LMS is one of the best known and most widely used robust estimators. The LMS method finds the residuals, $r_i = Y_i - \hat{Y_i}$, $i \in \{1, ..., n\}$ and n is the number of data points. As opposed to the LS method, which minimizes the $\sum_{i=1}^{n} r_i$, LMS solves the following nonlinear minimization problem:

$$\min med_i r_i^2 \tag{6}$$

It is well known that LMS is very robust to the outliers [16]; therefore, it is especially appropriate for our purposes where the data points, i.e. time sync. updates, could be coming from corrupted nodes and appear as outliers.

Finding the least median squares is a challenging optimization problem. There is no closed form solution for the optimization problem in equation 6, since we have to optimize the following [22]:

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

However, recently a randomized algorithm has been suggested that solves the LMS problem in time $O(n^d)$ and space $O(n)$, where d is the dimension of the data points [2]. Given $d = 1$ in our case, the amount of time and space required to solve 6 is feasible. In addition, there are a number of software packages that can perform this optimization, for example SAS/IML [22]. Porting these algorithms to a mote-class node is the subject of our current research.

VII. Conclusion and Future Work

Sensor networks have become prevalent in recent years, and the domain of their applications is being expanded as the technology advances. One of the fundamental tasks in sensor networks is the problem of time synchronization. Given the unattended nature of sensor networks, it is possible for an adversary to physically capture and subvert a subset of the nodes. By doing so, the adversary gains access to the network cryptographic keys and can participate in the ongoing communication without being detected. Therefore, as we argued in this paper, designing a secure time synchronization protocol is crucial to maintaining the functionality of the sensor networks. We described FTSP which is one of the major time synchronization protocols for sensor networks. The set of possible attacks on FTSP protocol was explained next. We outlined the adverse effects of the time synchronization attacks on some important sensor network applications, such as estimation. We then proposed a number of countermeasures to mitigate the effect of the security attacks. These proposals were mainly based on robust regression methods, i.e. the LMS method. Our goal is to provide a built-in, non-cryptographic mechanism by which the time synchronization protocol can correct for errors introduced by an adversary, i.e. an insider attacker.

We are currently implementing and testing those countermeasures on a sensor network testbed. Based on our results, we can evaluate the effectiveness of the 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] Bernholt, Thorsten. Computing the Least Median of Squares Estimator in Time $O(n^d)$. Proceedings of ICCSA 2005, LNCS 3480, pp. 697 706, February 2005.

[3] Coleri, S. PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks. M.S. Thesis, UC. Berkeley, December 2002.

[4] 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.

[5] 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.

[6] 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.

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

[8] Hohlt, B., Doherty, L., Brewer, E. Flexible Power Scheduling for Sensor Networks. Information Processing in Sensor Networks (IPSN), April 2004, Berkeley, CA.

[9] 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.

[10] Ledeczi A., Volgyesi P., Martoi M., et al. Multiple Simultaneous Acoustic Source Localization in Urban Terrain.

[11] 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.

[12] Oh, S., Russell, S., Sastry, S. Markov Chain Monte Carlo Data Association for General Multiple-Target Tracking Problems.

[13] Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J. D. SPINS: Security Protocols for Sensor Networks. Mobile Computing and Networking. Rome, Italy, 2001.

[14] Rmer,Kay. Time Synchronization in Ad Hoc Networks. Proceedings of MobiHoc 2001, ACM, October 2001.

[15] Manzo,M., Roosta,T., Sastry,S. Time Synchronization Attacks in Sensor Networks. Proc. of The Third ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN) November 2005, Alexandria, VA, USA.

[16] Rousseeuw, P.J. Least median of squares regression. Journal of American Statistical Association 79, 871880 (1984).

[17] 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.

[18] Sun,K., Ning, P., Wang, C., ”Secure and Resilient Clock Synchronization in Wireless Sensor Networks,” To appear in IEEE Journal on Selected Areas in Communications, Vol. 24, No. 2, February, 2006.

[19] 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

[20] Available on the web: www.xbow.com/Products/ productsdetails.aspx?sid=62

[21] Available on the web: http://www.cs.unc.edu/ tracker/media/pdf/ SIGGRAPH2001_CoursePack_08.pdf

[22] Available on the web: www.wabash.edu/econexcel/LMSOrigin/LMSIntro.doc