Secure Time Synchronization Protocols for Wireless Sensor Networks

Azzedine Boukerche*
SITE
University of Ottawa
Ottawa, CANADA
Email: {boukerch@site.uottawa.ca}

Damla Turgut
School of EECS
University of Central Florida
Orlando, FL, USA
Email: {turgut@eecs.ucf.edu}

Abstract— **Time synchronization is essential in wireless sensor networks as it is needed by many of the applications for basic communication. The inherent characteristics of sensor networks do not permit simply applying the traditional time synchronization algorithms. Therefore, many new time synchronization algorithms have been proposed and few of them have provided security measures against various degrees of attacks. In this paper, we review the most commonly used time synchronication algorithms and evaluate these algorithms based on factors such as their countermeasures against various attacks and the types of techniques used (cryptographic vs. statistical)**

I. Introduction

Wireless sensor networks naturally sense the desired event or phenomena in the real-world environment, communicate the sensed data to the global processing unit via the intermediate sensor nodes or gateway nodes for processing to draw relevant conclusions. Most of the time, the data received from multiple sensors are aggregated before reaching the final processing unit. In order to carry out the tasks discussed above, the physical time of the sensor nodes has to be synchronized with each other. The distributed wireless sensor networks heavily depends on the time synchronization for various reasons such as determining location and proximity of the deployed sensor nodes, intra-network coordination among different sensor nodes, temporal messege ordering, security, time division multiplexing in wireless communication, improving energy-efficiency of sensor nodes by scheduling the sleep times of the sensor nodes, and so on [16].

In the computer synchronization history, Lamport's work [11] is considered a pioneering approach that emphasizes the need of virtual clocks in computer systems in which causality is more important than the absolute time. Even though, the total ordering of the events was the focus of Lamport's method, this work has influenced the way the sensor networks have emerged today.

Most computer devices contain an internal clock, usually designed to be synchronized with the exact real-world time at the specific location of the computer. Although many functionalities depend on the clock even on desktop computers, for instance the scheduled Friday afternoon virus checks, or the popular "make" program, which determines whether a file needs to be recompiled comparing the timestamp of the source and object files. However in practice, a desktop computer can function correctly even if its internal clock is minutes or even years away from the correct time.

*Correspondant Author: Prof. A. Boukerche at boukerch@site.uottawa.ca

Let us first define the ways in which the clocks of two nodes A and B might be out of sync. Let us note the clock of a node X with a function $C_X(t)$, which returns the reading of the clock at real time t. The first type of difference is the *offset*: $\delta_{AB} = C_A(t) - C_B(t)$. That is the two clocks are identical, except that the clock of node A is early (if $\delta_{AB} > 0$). Now, we might fix this by setting the clock of node A back, however, that would create a problem, because the same time slice would appear twice for node A. This creates major problems for a number of protocols. It is better to set the clock of node B forward; however, most protocols simply require the nodes to keep track of their offsets without actually changing the internal clock.

The second type of synchronization difference is the *clock skew*, that is one of the clocks is running faster than the other. This can be expressed as a difference in the derivatives of the clock function in respect to the time: $\eta_A B = \frac{\partial C_A(t)}{\partial t} - \frac{\partial C_B(t)}{\partial t}$. While this appears to be a more difficult problem, if a node is aware of its clock skew, it can very easily account for it.

Neither clock offset nor clock skew requires periodic synchronizations. If we know the offset and the skew of a node's clock, we can calculate the time difference at any moment in time. However, the frequency of the clocks can change randomly because of environmental conditions such as temperature difference, or the aging of the hardware, a condition called *drift error*. The drift error appears as a non-zero second derivative in one or both clocks $\lambda_A B = \frac{\partial^2 C_A(t)}{\partial t^2} - \frac{\partial^2 C_B(t)}{\partial t^2}$. The clocks of sensor nodes usually accumulate several seconds of drift error per day, as the drift is not predictable, it needs to be solved using clock syncronization.

However, for sensor networks, that is, the correct synchronization of the clocks is frequently a necessary component of the ability of the sensor network to function correctly. Unsynchronized clocks can yield invalid observations, can create uncovered areas and timeslots, and in the worst case can disable the communication architecture of the network.

Let us consider several examples. The individual nodes of the sensor network are sending their timestamped observations to the sink.

In Figure 1, and intruder is sensed consecutively by sensors S_1 and S_2, and their reports sent to the sink. Based on the reports $(\text{intruder}, S_1, t_1)$ and $(\text{intruder}, S_2, t_2)$, knowing the locations of the sensors S_1 and S_2, and noticing that $t_1 < t_2$ and $t_2 - t_1 < 1$ second, the sink can correctly infer that the observations refer to the same intruder who is moving from left to right (in certain cases, this inference can be performed through in-network processing). However, this inference is valid only under the assumption that the clocks of the two sensors are synchronized at the level of tenths of seconds. Let us explain this further.

If the clock 1 and 2 are synchronized, that is, they have the same offset $\delta^{(1)} = \delta^{(2)}$ compared to a universal time t, then, $t_2^{(2)} - t_1^{(1)} = t_2 + \delta^{(2)} - t_1 - \delta^{(1)} = t_2 - t_1 > 0$. So, $t_2 - t_1$ indicates the correct order of the arrival to the sensors. However, if there is a large offset between these two, $\delta^{(1)} - \delta^{(2)} \ll 0$, that is, the clock of $\delta^(1)$ is early. We might have a situation that $t_2^{(2)} - t_1^{(1)} = (t_2 - t_1) + \delta^{(2)} - \delta^{(1)} < 0$, that is, the sink will infer incorrectly that the intruder is moving from right to left. This is case if the clock of the sensor S_1 is two seconds late, the inference would be that the intruder moves from right to the left. As a drift of several seconds per day is a normal occurrence for the internal oscillators of the devices, we can not rely on the initial setting of the clocks at deployment time. The clocks need to be synchronized periodically in the field.

Notice that the faster the intruder moves, the smaller $(t_2 - t_1)$ and the more accurate synchronization is needed in order to make the correct inferences.

Our second example concerns the wake-up time of the sensors. Sensors have limited power resources. To extend the lifetime of a deployed network, the sensor nodes are frequently selectively put to sleep. The idea of the method is that the set of currently active nodes at any given moment in time covers the area to be surveyed and form a connected network. If an attacker can modify the internal clock of certain sensor nodes, such that these nodes, for instance, do not wake up in time, certain areas might not be surveilled by the sensors for a certain amount of time, allowing an intruder to operate unreported.

Finally, Time Division Multiple Access (TDMA) based channel sharing protocols rely on the participating nodes to transmit at well defined time slots. Relatively small time drifts in the clock of the individual nodes can make the transmission intrude on the adjacent time slot, causing a collision. Repeated collisions can significantly disrupt the network. A detailed survey on clock synchronization protocols can be found in [17], [21].

The rest of the paper is organized as follows. The challenges and design issues of time synchronization protocols in sensor networks is summarized in Section II. In Section III, we survey some of the time synchronization protocol including possible attacks and proposed countermeasures against these attacks. The types of attackers and attacks are presented in Section IV. Section V gives detailed discussion about the approaches for secure time synchronization. We conclude in Section VI.

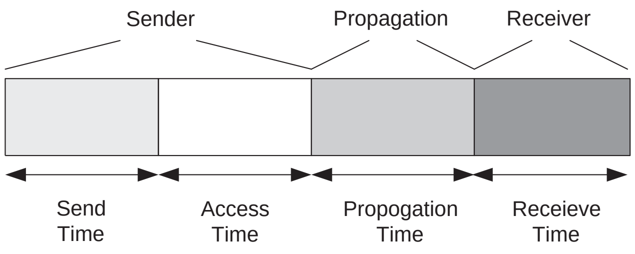

Fig. 2. Packet delay components

II. Time Synchronization in Sensor Networks

A. Challenges

Different applications will have different synchronization requirements and almost all the synchronization methods rely on message exchange between nodes. The non-determinism in the network operations can cause delays on the message delivery in turn contributes to the error in synchronization. Let us see how the source of a message's latency is decomposed in Figure 2. According to [9], [10], the time synchronization schemes have four basic packet delay components: send time, access time, propogation time, and receive time.

- **Send Time**. This is the time it takes to construct a message at the sender including the overhead of the operating system and the time to transfer the message to the network interface.
- **Access Time**. This is the delay encountered at the medium access control (MAC) layer prior to accessing the transmission channel due to contention, collisions, and so on. This delay is dependent on the MAC protocol in place. For instance, Carrier Sensing Multiple Access (CSMA) [8] requires every node to sense the carrier before transmitting, and does not start a transmission if the medium is busy. CSMA/CA consists of both carrier sensing and a collision avoidance; the IEEE 802.11 standard [7] is the best-known instance of CSMA/CA.
- **Propogation Time**. This is the time required in propagation of the message from sender to the receiver. The propagation time varies depending on the location of the sender and the receiver. For instance, if they are one-hop neighboring nodes in an ad hoc network, the propogation time equals to the physical propogation time of the message traveling in the media. On the other hand, the propogation time can be much larger if we add such as switching and queueing delays into the formula.
- **Receive Time**. This is the time for the receiver node to process the message and acknowledge the host the arrival of the message. Depending on the level in which the arrival time was timstamped, the receive time may or may not include the overhead of the transfering of the message from the network interface to the host.

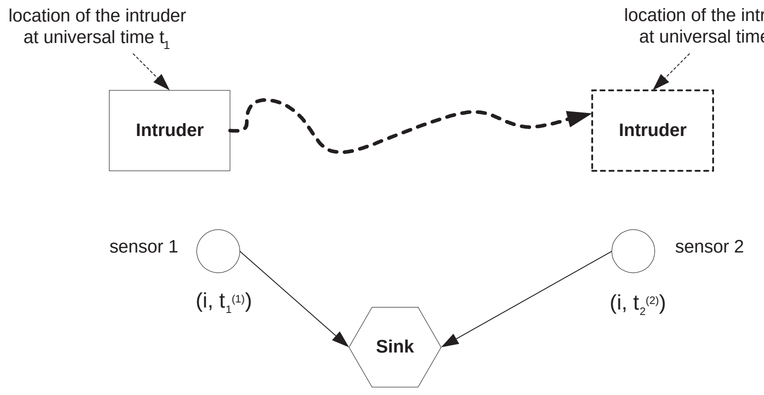
location of the intruder at universal time t_1
location of the intruder at universal time t_1
Intruder
Intruder
sensor 1
sensor 2
(i, $t_1^{(1)}$)
(i, $t_2^{(2)}$)
Sink

Fig. 1. Intruder movement detected by sensors 1 and 2 at times $t_1^{(1)}$ according to clock 1 and $t_2^{(2)}$ according to clock 2.

The challenge is not only the existance of packet delay but also the difficulty of prediction of the time needed on each delay components even though Figure 2 shows each component on equal length.

B. Design issues

Here, we present set of metrics for evaluating time synchronization protocols in sensor networks. Naturally, there are tradeoffs between these metrics; therefore, it is difficult to find a protocol which can satisfy them all.

- **Availability and scope**. All the nodes in the network can be synchronized based on a global time or a set of nodes locally in close proximity to each other can be synchronized based on a local time. Due to mainly the the energy and bandwidth constraints in large-scale sensor networks, global synchronization is not only difficult to implement but also can be costly as well. Only a few application would require global synchronization. Completeness of coverage within the specific region of nodes determines the availability requirement.
- **Cost and size**. Since most sensor nodes considered are small and low-cost devices, the secure synchronization algorithms must be designed to meet the cost and size requirement. For example, it is not a viable solution to attach a Global Positioning System (GPS) device to a sensor node.
- **Efficiency**. Due to the lightweight and small size of the sensor nodes used in many sensor network applications, they posses limited resources such as energy, memory, computation power, and so on. The security mechanisms developed within the time synchronization protocols must adapt to the limited computation power and memory of these sensor nodes since most security protocols are complex and requires extensive resources to run efficiently. Additionally, communication capabilities are also limited due to these resource constraints. GPS or Universal Time (UTC) are generally used to synchronize the network to an accurate time in traditional protocols such as Network Time Protoocol (NTP) [14]. Using GPS requires high energy consumption; therefore, it is not considered a viable solution. Reducing the energy consumption can be achieved by transmitting over sequences of hops of short distances rather than a single one-hop long path. The secure time synchronization algorithms should also take into consideration the time needed to synchronize the nodes.
- **Infrastructure**. Sensor networks may have to deployed in a random fashion to remote or dangerous regions. In this case, the sensor nodes are expected to self-organize themselves into a network since they cannot rely on an existing infrastructure such as NTP [14] in which the precise time is available to any node in the network from only a few hops away. NTP achieves this by providing the referenct time at various points in the network. The task of the secure time synchronization becomes complex due to the many factors such as scalability issues, mobility, and of course the lack of infrastructure to rely on.
- **Lifetime**. The duration of the synchronization can be one of the two forms: (i) almost instantenous or (ii) persistent in which case the synchronization lasts as long as the lifetime of the network itself. The nearly instantenous synchronization can be appropriate for applications in which an immediate action needs to be triggered based on the nodes' detection times of a specific event.
- **Network dynamics**. Irregardless of the type of deployment (random vs. pre-engineered) sensor nodes face high degree of network dynamics such as frequent changes in the network topology due to mobility, network partitioning, or energy depletion of the nodes. Designing secure time synchronization protocol becomes even more complex task when one needs to consider ways to ensure the running of the network operations smoothly under these dynamic changes.
- **Precision**. The level of precision or accuracy required generally depends on both the application and the objective of the synchronization. There are three basic types of synchronization methods. The first one relies on the ordering of messages and events; it is considered the simplest. The next method allows the nodes to keep track

of the drift and offset with respect to their neighboring nodes. This the most common type of synchronization encountered in the time synchronization protocol applications. The most strict and complex one is the global synchronization in which all the nodes are synchronized according to a global time accross the network. This method is the least used one since it is the most difficult to implement and also precise time synchronization is not always essential.

III. Secure Time Synchronization

These examples show us that time synchronization is vital for the correct operation of a sensor network. As relatively small time drifts can cause significant disruption, we cannot rely on the precision of the hardware; we need to use external synchronization protocols. Furthermore, it was found that as relatively small changes in the clocks can disturb the operation of the sensor network or even cause it to make erroneous inferences about the observed event, the time synchronization protocols are a convenient target for malicious attackers. Most time synchronization protocols were not designed with security in mind. Recently, however, several research groups performed an analysis of various vulnerabilities, and proposed countermeasures against them.

In the following, we survey some of the time synchronization protocols, discuss their benefits and outline possible attacks and proposed countermeasures.

A. Reference Broadcast Synchronization

A pioneering approach called *post facto synchronization* was proposed by Elson and Estrin [1]. This is low-power method of synchronization of clocks when the timestamps must be accurate for desired events. The nodes' clocks are are normally unsynchronized. When a phenomena is sensed, each node records the time of sensing according to its own local clock. Shortly after, a beacon node sends a synchronization message to the nodes within its transmission range. The nodes receiving this message can adjust their timestamps of the sensed phenomena to the time of the receipt of this synchronization message. The post facto synchronization forms the basis for the reference broadcast synchronization method.

A sender to receiver synchronization method is used in most time synchronization protocols. In this method, the sender transmits the timestamp information and the receiver synchronizes.

The Reference Broadcast Synchronization (RBS) [2] protocol differs from the sender to receiver synchronization since it uses receiver to receiver synchronization method. Basically, the RBS is based on a synchronization signal broadcasted by an external unit. The receivers record their local time when they received this reference message, and then exchange this information among themselves (see Figure 3). The recording of a message is not 100% exact, because of hazards such as the propagation time of the message or the processing time of the packet at the lower protocol layers. To improve the precision, a number of reference messages can be broadcasted, the nodes exchange the arrival times for each message and then find the best approximation using a least squares fit.

The keypoint of the RBS is that it uses broadcast technique within wireless medium to minimize latency and non-determinism issues related to latency in the time synchronization protocol. On the other hand, the most notable drawback of the RBS is its requirement of a network with a physical broadcast channel [2].

Let us consider a scenario shown in Figure 4 (a) where a group of nodes lie in the range of multiple broadcast instead of a range of a single broadcast. Both sender nodes A and B send a synchronization message. According to the transmission ranges, nodes 1-4 will be able to synchronize with node A and nodes 4-7 can synchronize with node B; however, nodes A and B cannot hear each others synchronization message. Among the nodes, node 4 is the only one that can hear the synchronization messages from both nodes A and B. This means that node 4 can correlate the clocks in node A's neighborhood to the clocks in node B's neighborhood or vice versa. In Figure 4 (b), we have 3-hop network topology and the corresponding logical topology in Figure 4 (c). The dotted lines, in other words the graph edges, are drawn between two nodes whose clocks are known to each other, for instance, node pairs (1-4),(7-9),(8-9), and so on. Note that there are two links between nodes 8 and 9 meaning that they have two receptions in common since they are located in the overlapping region of C and D.

Possible attacks on RBS: In RBS, two nodes upon receiving a broadcast signal, exchange their local clock time. An attack can happen if one of the receiver nodes is compromised with an incorrect time. The compromised node then can send the incorrect time information to its neighbor causing the uncompromised node calculating an incorrect offset.

Multi-hop version of RBS can face attacks as well. If compromised node is located in any of the overlapping regions, it can inject an incorrect value into the clock conversion process, affecting multiple regions at once. This miscalculation in the clock conversion can be propagated across the network.

B. Time Synchronization Protocol Sensor Networks (TPSN)

TPSN [6] has two phases: *level discovery* and *synchronization*. In the level discovery phase, a spanning tree is created for the sensor network where each node is assigned a level. The root of the tree is usually a base station and assigned level 0. It is assumed that a node at level n can communicate with a node or a set of nodes at level $n-1$. In the synchronization phase, the child nodes are synchronized to the parent. The synchronization is initiated by the child node which sends a synchronization packet at time t_1. This is received by the parent at t_2 and an acknowledgment packet is sent in response at time t_3. The values of $t2$ and $t3$ will be included in the acknowledgement packet. This packet is received by the child node at time t_4 (see Figure 5). Knowing these four time values the child node can calculate its clock offset relative to the parent node as well as propogation delay as follows:

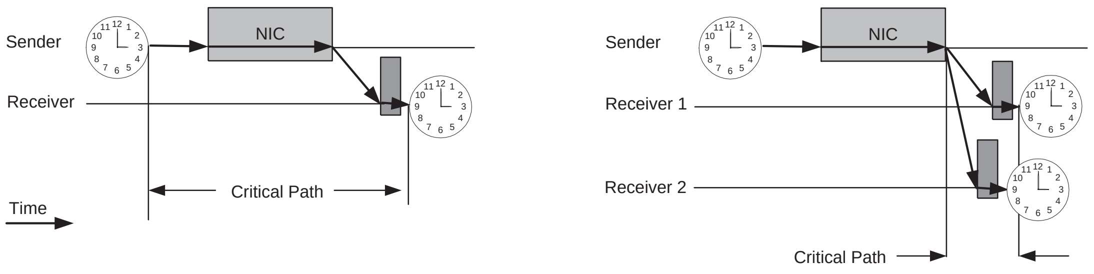

Fig. 3. A critical path analysis for traditional time synchronization protocols (left) and RBS (right) (adapted from [2]).

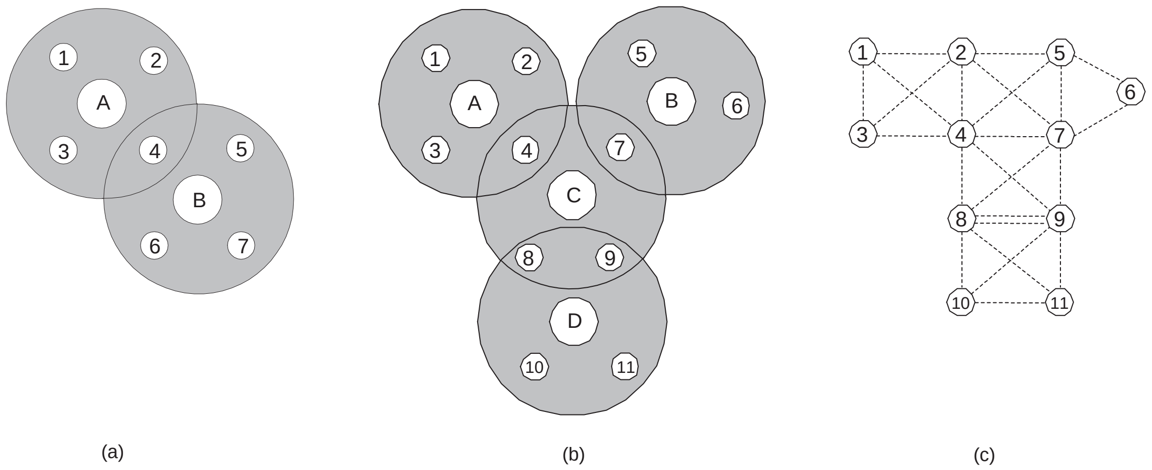

Fig. 4. Simple (a) versus complex (b) multi-hop network topology with the corresponding logical topology (c) of the complex topology (adapted from [2]).

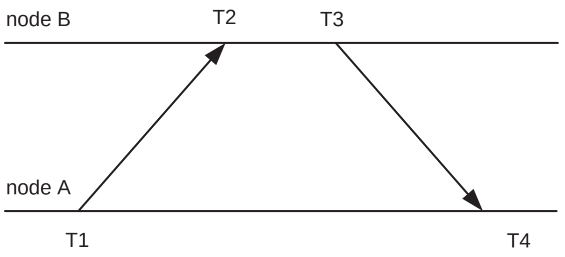

Fig. 5. Two-way message exchange between pair of nodes. T2 and T3 are measured in node B's clock while T1 and T4 ar measured in node A's clock (adapted from [6]).

$$\Delta t = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{1}$$

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2}$$

The decomposition of the packet delay has been depicted in Figure 6 similar to [9], [10] as shown in Figure 2. The transmission and reception time are more detailed here. The transmission time is the time it takes to transmit a packet on a bit-by-bit basis at the physical layer via the wireless link [6]. The reception time is the time required to receive and forward all the bits to the link layer. Both transmission time and reception time are generally considered deterministic; however, variations can occur depending on the underlying hardware structure.

The nodes are assumed to have unique ids and each node has knowledge of its neighboring nodes. TPSN takes advantage of only the symmetric links for pair wise synchronization between nodes even though the network may also have asymmetric links. This can be considered one of the drawback of the protocols. Other drawback may include its limited suitability for applications serving highly mobile networks and its lack of support for multi-hop communication. On the other hand, TPSN is scalable and its computation overhead is less than some other protocols such as NTP [14]. The root selection and the tree construction mechanisms need to be reinvoked when topology changes occur due to node failures or other factors.

Possible attacks on TPSN: A compromised node cannot cause any problem by requesting a time synchronization message because the message will reach to the parent node only. On the other hand, the compromised node can send erroneous time information to its children.

Naturally, the child node is relying on the parent for its clock synchronization, by providing incorrect values for t_2 and t_3, the parent can set an arbitrary offset on its child node. What is more, this incorrect offset will then be propagated down the tree. Therefore, the number of nodes whose synchronization

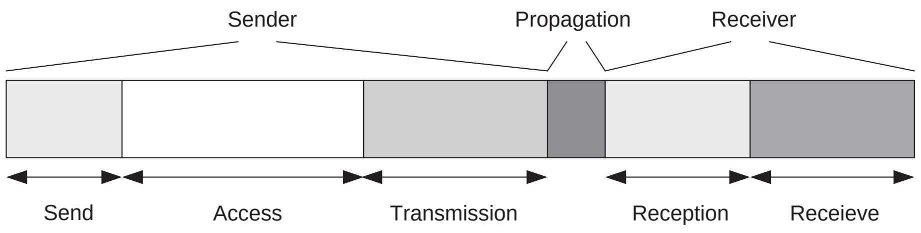

Fig. 6. Decomposition of packet delay over a wireless link in TPSN (adapted from [6]).

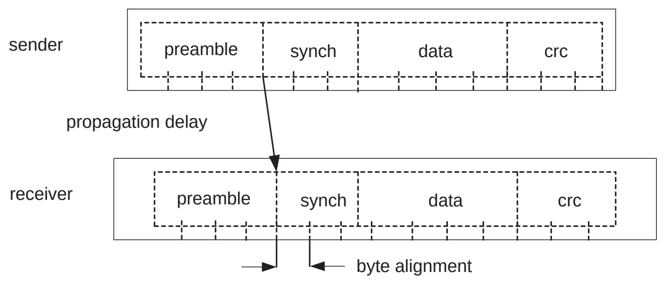

Fig. 7. Data packets transmitted over the radio channel (adapted from [13]).

can be affected by the compromised node depends on the location of the compromised node on the tree. One way for a malicious attacker to compromise a larger number of nodes is to reposition itself in a higher location on the tree or to answer queries instead of the proper parent. This is surprisingly easy to do in the original algorithm.

Yet another type of attack can surface when the compromised node misinforms its level in the tree, basically announcing a lower level than its current level. The compromised node can also attempt to trick other nodes at its level in requesting synchronization updates from itself. Furthermore, the compromised node can disconnect a number of nodes from being included in the tree by simply not participating in the level discovery phase.

C. Flooding Time Synchronization Protocol (FTSP)

In FTSP [13], the nodes are participating in a process in which a *root node* is elected. The root is the origin of the time synchronization messages. If a node does not hear a time synchronization message for a while, it declares itself the new root. The protocol requires that if at a later time the node receives a time synchronization message from a node with a lower id than itself, it gives up its root status. When a node receives a time synchronization message from the root, it adjusts its clock and broadcasts its own time to its neighbors. In the message broadcast, the preamble bytes are transmitted first, followed by *sync* bytes, message descriptor, the actual message, and finally *crc* bytes (see Figure 7).

Possible attacks on FTSP: The FTSP protocol is more robust for node failures than the TPSN protocol, as there is no need to maintain a tree structure which is notoriously vulnerable to single point failures - the failure of a single node can disconnect the whole subtree. The weak point of the FTSP protocol is the election process. Any node can declare itself a root, and the protocol relies on the node to step back if a lower id root appears. A compromised node can easily masquerade as a root, and by declaring a very low id it can actually dislodge the existing legitimate root. Then, by sending a synchronization message with a fake timestamp, it can make the nodes synchronize to an incorrect time.

D. Countermeasures for the attacks

In this section, we describe the possible countermeasures for time synchronization attacks in both single-hop and multi-hop networks [12]. In single-hop networks, transmission range of each node reaches to every other node in the network. Let us consider a specific case of a single-hop networks in where there is a base station and its transmission range covers all the nodes in the network. The challenge for this type of networks is to preclude the malicious node(s) from compromising the base station and immediately start injecting the network with invalid timing information. In this case, the message from the base station must be authenticated in order to carry out the correct sequence of time synchronization methods. This can be achieved by utilizing a broadcast authentication scheme such as μTESLA [15]. Another approach can be the use of different private keys between the sender and receiver nodes.

In the multi-hop networks, many nodes may need to communicate with each other via intermediate nodes due to the limited transmission ranges. It makes sense for nodes to receive the global time of their immediate neighbors instead of neighbors several hops away. In that case, we need to take into consideration the incurred network delays between these nodes and their far away neighbors. An approximation approach can be used find an upper bound on the error produced by the malicious node. Introduction of redundancy to the network is another viable approach. For instance, both FTSP and TPSN compute the offset and skew of their clock based on the timing information obtained from only one neighboring node. This redundancy approach can be easily applied to FTSP where a set of nodes can be used for the synchronization computations. The nodes can take the median of these received multiple updates. The private keys can be setup between the nodes and their neighbors at the beginning such that a malicious node can be precluded from injecting erroneous updates into the network. Furthermore, if a node becomes aware of one of its neighbors update value being substantially different than the updates from its other neighbors, the node can refrain itself from including the updates from the suspicious neighbor into its computations of clock skew and offset. This approach of containment works however under the assumptions that the nodes have sufficient number of sources for the updates. For instance, in TPSN, children of the nodes nearby the malicious or compromised node must find another parent which may not always be feasible. Essentially, maintaining multiple trees come with a price whereas in FTSP, no such cost exist. Utilizing the LS linear regression by each node in the network to compute the skew of its clock can further enhance security in time synchronization protocols. Algorithms such as RANSAC [3] can be used for this purpose.

IV. Attackers and attacks

The possible attacks against time synchronization protocols depend on the nature and capabilities of the attackers. We will first identify three different types of attackers, outline the types of attacks they are capable of, and then we will discuss the various types of defenses proposed against these types of attacks.

We will describe the system with the characters regularly used in the description of cryptographic protocols. We assume that Alice and Bob (and potentially, additional nodes Carol, Dave, and so on) are engaging in a time synchronization process.

The **malicious outsider** Malory is a wireless device inserted in the range of the nodes of the sensor network, which has the ability to send and receive packets. We assume that the attacker can eavesdrop on any ongoing transmission, we can also assume that the attacker can eavesdrop on any ongoing transmission, we can also assume that the attacker can transmit messages which are physically indistinguishable from the other nodes messages. However, this type of attackers do not have access to keys or other confidential information, other than what it can infer from eavesdropping on transmissions.

Attacker with jamming and replay ability, Jimmy is an attacker which has the ability to jam the message, record it and possibly replay it at later time. This type of attack is called a *pulse delay attack*. Although the existence of jamming in principle can be detected, it requires significant resources, and by default, most nodes are not prepared for it.

A **compromised node** is a node which was taken over by the attacker. We will denote them with Zach (for zombie). One example of this is the physical capture of the node by an attacker, although a node can, in principle, be compromised with purely software methods. Compromised nodes have access to all the keys and other information of the original node, and represent the most difficult type of attackers to defend against.

The challenge of secure time synchronization is to defend against all the three types of attackers. An attacker is considered successful if it succeeds in making the nodes calculate an incorrect offset. By default, all three time synchronization protocols we described are vulnerable to all three types of attackers.

V. Approaches for secure time synchronization

Malicious outsiders can affect all types of protocols. The primary defense against a malicious outsider is cryptographic techniques of the authentication of messages. If the sender and the receiver is sharing a key K_B, they can use it to sign the messages. To prevent an attacker to capture a valid message and insert a copy of it later in a different synchronization round, the sender sends a random *nonce* in the initial message, which then needs to be signed by the synchronization partner. While the attacker can still replay the same message in the same synchronization round, by simply considering only the first arrived message, the receiver can ignore the malicious outsider.

Ganeriwal et. al. [4], [5] proposed a series of secure time synchronization protocols based on this idea. The protocols are adapted to pairwise single-hop, multi-hop synchronization and for group synchronization. The protocols can also detect the existence of a pulse delay attack by calculating the end to end delay d of the message. If the delay is larger than a predetermined threshold $d*$ the protocols assume that an attack is in progress and abort the synchronization. We should note that key exchange is a major problem for these types of algorithms, due to the ways in which sensor nodes are deployed, which does not always permit the exchange of the keys in a secure environment.

Notice that cryptographic methods are not feasible against a compromised node, which has all the keys and knowledge to correctly answer all the challenges, appropriately sign its messages. If the time synchronization protocol happens only between Alice and Zach, it is impossible for Alice to detect or mitigate the attack. However, for protocols with a larger number of participants, we can use the redundancy in the synchronization messages to identify the malicious participants or messages. We note that delay attacks can be performed by either Zach or Jimmy, but not Mallory.

Song et al. [18] propose a method for making time synchronization protocols resilient to delay attacks based on techniques of *outlier detection*. The essential assumption behind this method is that the synchronization signals received from compromised nodes will be "much different from other". Thus, messages coming from compromised nodes can be identified using statistical techniques as outliers, eliminated from the package, and the synchronization performed with the remaining nodes.

The authors propose two alternative methods. One of them uses the generalized extreme studentized deviate (GESD), a generalization of the well known Grubb's test from statistics. GESD can identify multiple outliers in a sample drawn from a normal distribution. GESD requires as one of its outputs the estimated number of malicious nodes.

A somewhat simpler approach is based on a delay threshold. At the setup time of the system, the nodes determine the maximum amount of time offsets they will tolerate, based on information about the typical drift rate of the nodes. A received offset which is higher than this value will be considered to come from a malicious node and discarded.

As an observation, naturally, Zach, the compromised node would have exact knowledge about the thresholds used (but not Jimmy). Therefore, Zach has the possibility to remain undetected, by setting the delay such that it will put it just below the threshold (or, in the GESD case, such that it will not be identified as an outlier), but still have a distorting effect on the time synchronization process. Thus, the statistical techniques can only reduce, but not necessarily eliminate the effect of delay attacks by nodes with insider knowledge.

Sun et al. [19] propose a statistical method for secure and resilient clock synchronization in the presence of compromised nodes. The techniques are applied for both *level-based clock synchronization*, where a hierarchical structure of nodes is developed which determines which node is synchronized with whom, and diffusion based clock synchronization, which does not use such a structure and simply relies on the reachability information of the network. Naturally, the level based approach allows for a more disciplined control of the synchronization flow, and thus a higher accuracy, whereas the diffusion method has the advantage that it can be applied to dynamic sensor networks with mobile nodes.

Furthermore, the authors consider both the case with a single source of synchronization information, and with multiple sources. For instance, in the single source case, the goal is assumed to be to find the clock offset δ_{iS} from the node to the source. The technique assumes that at every level, a normal node collects $2t+1$ candidate source clock differences from its $2t+1$ neighbors, and chooses the *median* of them. Thus, the node can tolerate up to t compromised nodes, while retaining correct synchronization. Similar considerations apply to the diffusion based approach. In the case of multiple sources, the node can receive synchronization information from $2s+1$ sources, synchronized to the same external standard (such as a GPS signal) and tolerate up to s compromised sources by selecting the median.

Note that this approach uses the whole redundancy of the system to defend against an external attack - out of $2t+1$ recorded offsets, the method will pick a single one, the offsets median. Approaches which assume a benign environment usually select the *mean* of these measurements, therefore improving accuracy - however, the mean is vulnerable to even a single malicious node.

In addition, this approach requires a unique pairwise key based authentication of the nodes. Otherwise, the malicious node could impersonate multiple nodes (the so-called Sybill attack).

A significantly improved version of this technique was presented in [20]. In the approach called TinySeRSync, time synchronization is performed in two phases. While we will call them Phase I and Phase II for convenience, these two processes are taking place asynchronously in the sensor network. In the first phase, single-hop pairwise synchronization is performed. The main feature of the pairwise synchronization process is that it relies on a hardware enhanced authenticated MAC layer timestamping. The hardware is programmed to add a timestamp authenticated with a message integrity code (MIC) to every MAC packet transmitted. This is especially challenging for the newer radios, such as the ones on the newer generation MICAz motes, where the time required to authenticate the timestamp can interfere with the transmission rate of the radio. The authors propose a prediction based approach where the authenticated timestamp includes a prediction of the time required to calculate the MIC.

Through these techniques, the nodes achieve a sufficiently good local level synchronization, which is exploited in the second phase. The second phase implements a global synchronization using the μTESLA broadcast authentication protocol. This protocol relies on the loose time synchronization between the nodes, uses a unidirectional keychain. Messages received need to be stored by the receiver and will be authenticated only after several timeslots. This prevent an attacker from forging messages, but it opens the doors for a denial of service attack. The messages received need to be buffered for future authentication, and as the memory of sensor nodes is limited, Mallory can create fake messages, which will not pass the authentication test, but will fill the buffer, preventing the node from receiving legitimate messages. To prevent denial of service attacks, the authors propose a modified version of μTESLA. To reduce the timeslots when the adversary nodes can flood the node with messages based on captured keys (which the receiver node needs to store for future authentication). TinySeRSync uses an implementation with very short delays r (made possible by the good local synchronization achieved in phase I). However, such short delays would require the generation of a large number of keys, the implementation alternates short intervals r used for message broadcasting, with long intervals R used for broadcasting the disclosed keys.

The global synchronization in TinySeRSync still relies on the selection of the median from the $2t+1$ candidate offsets, therefore tolerating the presence of at most t compromised nodes.

Notice that the approach presented in [19], [20] uses the *median* rather than the *mean*, as a choice of the estimated time offset, thus obtaining a high protection against malicious nodes (provided the technique is coupled with cryptographic defenses). However, it sacrifices the ability to improve precision through multiple independent observations.

We can attack the general problem of finding the best estimation of the time offset δ_{best} from a set of candidate offsets $\{\delta_1, .., \delta_n\}$ by applying the principles of *robust estimation*. We note that the individual offset measurements δ_i might have natural noise, but some of them might be a result of a malicious attack. For any estimation method, the *breakdown point* is the smallest number of contaminated values which can move the estimate arbitrarily far from the correct value. Unfortunately, the most frequently used estimators, the average and the least squares estimator have a very low breakdown point; a single malicious value can modify the estimate arbitrarily far. Manzo et. al [12] propose the use of the Least Mean Squares (LMS) estimator for a more robust modelling. The generalized extreme studentized deviate (GESD) used by

TABLE I

A CONCISE SUMMARY OF THE VARIOUS TECHNIQUES PROPOSED FOR SECURE TIME SYNCHRONIZATION

Approach	**Protects against**	**Uses crytographic techniques?**	**Uses statistical techniques?**
Ganeriwal et. al. [4]	Jimmy, Mallory	Yes	Yes
Song et. al. [18]	Zach, Jimmy, Mallory	No	Yes
Sun et. al. [19]	Zach	No	Yes
TinySeRSync, Sun et. al [20]	Zach, Mallory	Yes	Yes
Manzo et. al. [12]	Zach	Yes	Yes

[18] is another example of the application of the techniques of robust estimation.

VI. CONCLUSIONS

Among the many challenges in designing and employing wireless sensor networks is the clock synchronization between the sensor nodes. Agreeing on a common time is needed and even required by many of the sensor applications to carry out the sensing, communication, and processing of the sensed data. The time synchronization protocols in traditional wired networks cannot simply be reused in the wireless sensor networks domain due to the inherent characteristics and limited resources of these networks. Therefore, several time synchronization protocols have been proposed recently; however, most of them do not consider security aspect during the design stages. There are only handful protocols where the security has been taken into the consideration.

In this paper, we reviewed three most common secure time synchronization protocols: i) Reference Broadcast Synchronization (RBS), ii) Time Synchronization Protocol Sensor Networks (TPSN), and iii) Flooding Time Synchronization Protocol (FTSP). We then evaluated these algorithms based on factors such as their countermeasures against various attacks and the types of techniques used (cryptographic vs. statistical).

REFERENCES

[1] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–6, April 2001.

[2] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 147–163, December 2002.

[3] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of ACM*, 24(6):381–395, June 1981.

[4] S. Ganeriwal, S. Capkun, C.-C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *Wireless Security Workshop (WISE)*, pages 97–106, September 2005.

[5] S. Ganeriwal, S. Capkun, and M. B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and Systems Security*, March 2006.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *ACM Proceedings of the 1st international conference on Embedded networked sensor systems (SenSyS 2003)*, pages 138–149, November 2003.

[7] IEEE std 802.11b-1999. part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, August 1999.

[8] L. Kleinrock and F. A. Tobagi. Packet switching in radio channels: Part I - carrier sense multiple-access modes and their throughput-delay characeteristics. *IEEE Transactions on Communications*, COM-23(12):1400–1416, 1975.

[9] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, C-36(8):933–939, 1987.

[10] H. Kopetz and W. Schwabl. Global time in distributed real-time systems. Technical report 15/89, Technische Universität Wien, 1989.

[11] L. Lamport. Time, clocks, and ordering of events in a distributed system. *Communications of ACM*, 21(4):558–565, July 1978.

[12] M. Manzo, T. Roosta, and S. Sastry. Time synchronization attacks in sensor networks. In *The Third ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2005)*, pages 107–116, November 2005.

[13] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *ACM Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSyS 2004)*, pages 39–49, November 2004.

[14] D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1483, 1991.

[15] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. Spins: security protocols for sensor networks. In *International Conference on Mobile Computing and Networking*, pages 189–199, July 2001.

[16] K. Römer. *Time Synchronization and Localization in Sensor Networks*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), Zurich, Switzerland, 2005.

[17] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: A survey. *IEEE Network Magazine's special issue on Ad Hoc Networking: Data Communications & Topology Control*, 18(4):45–50, July/August 2004.

[18] H. Song, S. Zhu, and G. Cao. Attack-resilient time synchronization for wireless sensor networks. *Ad hoc Networks Journal, Elsevier*, 5(1):112–125, January 2007.

[19] K. Sun, P. Ning, and C. Wang. Secure and resilient clock synchronization in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 24(2):395–408, February 2006.

[20] K. Sun, P. Ning, C. Wang, A. Liu, and Y. Zhou. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *To appear in Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, November 2006.

[21] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, May 2005.