CLOCK SYNCHRONIZATION FOR MULTIHOP WIRELESS SENSOR NETWORKS

BY

ROBERTO SOLIS ROBLES

Ingen, Instituto Tecnologico de Zacatecas, 1992
Maest, Instituto Tecnologico y de Estudios Superiores de Monterrey, 1999

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Doctoral Committee:

Professor P. R. Kumar, Chair and Director of Research
Professor Lui Sha
Professor Carl A. Gunter
Assistant Professor Yih-Chun Hu

ABSTRACT

In wireless sensor networks, more so generally than in other types of distributed systems, clock synchronization is crucial since by having this service available, several applications such as media access protocols, object tracking, or data fusion, would improve their performance. In this dissertation, we propose a set of algorithms to achieve accurate time synchronization in large multihop wireless networks.

First, we present a fully distributed and asynchronous algorithm that has been designed to exploit the large number of global constraints that have to be satisfied by a common notion of time in a multihop network. For example, the sum of the clock offsets along any cycle in the network must be zero at any instant. This leads to the concept of "spatial smoothing." By imposing the large number of global constraints for all the cycles in the multihop network, these time estimates can be smoothed and made more accurate.

The algorithm functions by simple asynchronous broadcasts at each node. Changing the time reference node for synchronization is also easy, consisting simply of one node switching on adaptation, and another switching it off. It has been implemented on a Berkeley motes testbed of forty nodes, and comparative evaluation against a leading algorithm is presented.

Next, considering that most of the clock synchronization protocols that have been developed do not provide means to detect security attacks which could render them useless, we present a secure network-wide clock synchronization protocol. At the same time, this protocol allows the nodes to securely discover the network's topology by

detecting and isolating all links that have fallen under the control of attackers. The protocol detects the attacks using only timing information under certain conditions. It has been implemented on an IMote2 testbed of twenty five nodes. Experimental results are provided.

To my beautiful wife, Sahara, and my wonderful children, Sahara and Alberto.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the help, encouragement, and support of many people.

I am specially grateful to my adviser, Professor P. R. Kumar, for his guidance and continued support throughout the years. Without him, this work would have not been possible. I thank the members of my Ph.D. committe, Professors Lui Sha, Carl Gunter and Yih-Chun Hu for their invaluable comments and suggestions to improve this dissertation. I would also like to thank the staff of the Coordinated Science Laboratory and all my friends and colleagues there.

I want to acknowledge the financial support received from Instituto Tecnológico de Zacatecas, Universidad Autónoma de Zacatecas, Fulbright-CONACYT, PROMEP, and Professor P. R. Kumar, which allowed me in one time or the other, to complete this dissertation.

Finally, thanks to my parents, Honoria and Rufino, for always being there for me, and specially to Sahara, my wife, who endured this long journey, and has given me the joy of two beautiful children, Sahara and Alberto; they all have made it worth the effort.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Wireless Sensor Networks

Technological advances made in recent years in the fields of microelectronics and wireless communications have enabled the integration of sensors, actuators and radios, which has lead to the development and deployment of wireless sensor networks. Wireless sensor networks have grown in popularity and the applications that can be developed for such networks are widespread, ranging from target tracking surveillance [2, 3], to biomedical health monitoring [4] and seismic sensing [5]. Along with these applications come new design challenges [6, 7], which stem from the fact that a wireless sensor network has limited resources in terms of memory, computation power, bandwidth and energy; and design constraints based on the monitored environment. These play a key role in determining many aspects of the network such as size, deployment scheme and topology. For example, in terms of resource constraints, the MICA2 Berkeley motes [8] use an 8-bit micro-controller running at 4 MHz with a limited instruction set, have only 4KB of RAM memory and can transmit up to 19.2 kbps. They are battery powered by an inexpensive pair of AA batteries that produce between 2.0 and 3.2 V, and the lifetime is mainly determined by its awake duty-cycle and radio usage.

1.2 Importance of Clock Synchronization

For several distributed applications either it is necessary for the processors to have a common notion of time, or to have it would improve their performance. Examples of these applications are authentication protocols, media access protocols, and database consistency [9]. In the case of wireless sensor networks, applications such as object tracking, environmental monitoring, TDMA scheduling, and data fusion, require some kind of timing service in order to determine the order in which events have unfolded and also the actual times of the events themselves. The synchronization would also allow the nodes to save energy. In wireless sensor networks, the accuracy of object localization or tracking is limited by the accuracy of clock synchronization [10]. When actuation is also performed over the network, accurate clock synchronization is needed to avoid delay induced instabilities, and improve control system performance [11]. Due to its importance, several algorithms have therefore been designed ([12–30]) in order to synchronize the clocks of a distributed system over traditional networks, and also over wireless sensor networks.

In this dissertation, two algorithms to achieve clock synchronization in a multihop wireless sensor network are presented as well as the results on their performance in implementations.

For the first algorithm, the key novelty of the approach presented consists of a new notion of "spatial smoothing." It exploits global constraints imposed by a common notion of time to improve the performance of clock synchronization, and yet achieves this through a completely asynchronous, distributed algorithm. The heart of the idea is the following. Suppose O_{ij} is the offset of the clock at node j with respect to the clock at a neighboring node i, at a certain time. Then an estimate of $\widehat{O}_{ij}$ can be formed by bilateral exchange (or broadcast) of timestamped packets between the neighboring nodes i and j. These estimates are, however, noisy and

are based only on the particular timestamped packets exchanged between the two neighboring nodes. Now consider any cycle formed by nodes $i_1, i_2, \ldots, i_n, i_{n+1} = i_1$, in the multihop network. Necessarily, $\sum_{k=1}^{n} O_{i_k i_{k+1}} = 0$ must be satisfied by the very notion of common time. This is an example of one global constraint, and there are several such global constraints, one for every cycle in the graph. By taking advantage of these constraints, the noisy estimates $\widehat{O}_{ij}$ can be further smoothed to give better estimates. Imposing these global constraints can thus improve the time estimate at every node in a multihop network with respect to any chosen reference node. Essentially, this procedure allows full exploitation of all bilateral estimates, i.e., all global information, to synchronize the clock at every node, thus making possible the spatial smoothing of otherwise noisy local time estimates.

Thus we obtain a completely distributed asynchronous algorithm, where nodes communicate only with their neighboring nodes, to achieve these global constraints. Our distributed algorithm takes advantage of all estimates of bilateral offsets all over the network to improve performance at every node, and not just those, say, along a rooted tree. The incorporation of such a large number of global constraints improves clock synchronization, especially in large networks where there are indeed a large number of such constraints.

As noted above, this algorithm does not require any constructions such as a rooted tree. In fact, it does not need any global topology knowledge. Instead it only uses asynchronous distributed broadcasts at each node. Moreover, switching the time reference from one node to another is also easy. It simply consists of one node switching on adaptation, and another switching it off, and the entire network then adapts to the change.

The results of an implementation over a 40-node Berkeley motes testbed and a comparison of these results with a leading clock synchronization protocol on the testbed are presented.

For the second algorithm, security is taken into account and based on theoretical work by Chiang et al. [1], we first develop and implement a secure clock synchronization protocol over a single link that is able to detect man-in-the-middle attacks using only timing information. In man-in-the-middle attacks, the attacker intercepts messages exchanged between two nodes and relays such messages in a way that makes the two nodes believe that the link they share is valid, while in reality the characteristics of the link have been modified. For instance, delays are introduced which could affect the clock synchronization service. By using the secure clock synchronization protocol, nodes are able to impose restrictions on the kind of delays a man-in-the-middle attacker can add to the packets exchanged between them. We validate the assumptions and properties of the theoretical protocol by means of an implementation performed using IMote2 motes and TinyOS 2.1. We further proceed to successfully implement a mechanism to detect half-duplex man-in-the-middle attackers.

Finally, the secure clock synchronization protocol is extended to securely synchronize the clocks in a multihop network. At the same time this protocol also allows the nodes in the network to securely discover the topology of the network. The ultimate goal of this secure network-wide clock synchronization protocol is to obtain a network-consistent clock.

This protocol is able to detect misbehaving or compromised links, disseminate information regarding those links and effectively isolate them. It is divided into four steps: single link check, neighborhood check, network consistency check, and removal of network inconsistencies.

The protocol has been implemented on a testbed comprised of 25 Crossbow IMote2 sensor nodes on top of TinyOS 2.1. Experimental results are presented.

CHAPTER 2

MOTIVATION AND RELATED WORK

The applications deployed over wireless sensor networks range from environmental monitoring [5,31–34], health and wellness monitoring [35,36] distributed control [37], and object tracking [10,38], to control over networks [39–41]. Most of these applications need to determine the times of the events that occur during their execution, thus requiring a clock synchronization service.

For instance, in [10], a network of directional sensors is intended to monitor a region which is crossed sporadically by objects assumed to be moving at nearly constant velocity. Using the times at which sensors detect objects crossing their "field of vision," the trajectories of the objects can be determined as well as the sensor directions, which are also unknown a priori.

An accurate notion of time can also be used to improve communication network performance. By having nodes in a wireless network synchronized, implementation of MAC protocols such as TDMA or SEEDEX [42] would be feasible. This can lead to greater throughput [43]. In addition, it can also be used to conserve energy by reducing collisions when nodes transmit over the shared medium, as well as sleep while awaiting their turn to transmit. Accurate clock synchronization can allow nodes to more accurately coordinate their sleep states by employing advanced duty-cycling schemes [44].

Clock synchronization has been an important topic of research for several years. Several algorithms have been developed, mainly for traditional wireline networks, and, lately, many for wireless sensor networks. This chapter presents the most important

references in this field beginning with the seminal work by Lamport on virtual clocks, followed by some important theoretical results and the main algorithms proposed for traditional, ad-hoc, and sensor networks.

2.1 Virtual Clocks

The seminal work in clock synchronization is due to Lamport [45], where the idea of virtual clocks is used. In his system, he defines the "happened before" relation (denoted by the symbol $\rightarrow$) as follows:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.
4. Two distinct events a and b are said to be "concurrent" if neither $a \rightarrow b$ nor $b \rightarrow a$ hold.

Also, a *logical clock* C_i is defined as a function that assigns a real number $C_i(a)$ to any event a in that process. The clock meets the following condition: For any events a, b in process i, if $a \rightarrow b$, then $C_i(a) < C_i(b)$ as long as the following holds:

1. If a and b are events in process P_i and a comes before b, then $C_i(a) < C_i(b)$.
2. If a is the sending of a message by process P_i and b is the receipt of the same message by process P_j, then $C_i(a) < C_j(b)$.

Finally, a total ordering "happened before" relation, denoted by $\Rightarrow$ is defined. It extends the $\rightarrow$ relation to a total ordering by assigning processors a priority to break the ties. However, the resulting total order is not unique.

This work also determines that in case of physical clocks, the closest that two clocks can get is approximately $d(\rho\tau + \epsilon)$, where d is the diameter of the network, ρ is the drift rate of the clock, τ is the time interval over which a clock synchronization message is sent, and ϵ is the maximum unpredictable delay of the message.

2.2 Theoretical Results

Several theoretical results have been published regarding clock synchronization, the most important being the following:

- In [46], Dolev and Halpern prove the impossibility of achieving clock synchronization if more than a third of the nodes in the network are faulty, and give the requirements to be met in order to achieve it.

- In [47], Lundelius and Lynch show that no algorithm can synchronize clocks exactly, and provide lower and upper bounds on how close the clock values can be at the same real time in a fully connected network.

- In [48], Biaz and Welch provide a lower bound for the optimal clock synchronization achievable for any network topology, taking into account the diameter of the network, and provide a tighter bound for a specific class of network topologies.

- In [11], Graham and Kumar show that it is fundamentally impossible to estimate time between two clocks precisely without assuming symmetric delays. Clock skew can be estimated regardless, and also, after two nodes have clock synchronized, when one node sends a packet to the other, the sender can perfectly predict the time at which the packet will be received in terms of the receiver's clock.

- In [49], Gurewitz et al. propose a method to estimate the one-way delay for each link in an N-node network that works by summing up all single-hop one-way measurements made along various cyclic paths. They show that the maximal number of independent cyclic path delays that can be computed is smaller than the number of links by N-1, which means that it is not possible to uniquely determine the one-way delays.

- In [50], Freris and Kumar analyze networks of clocks, and characterize what is determinable from what is not.

2.3 General Network Algorithms

The basic idea of a clock synchronization algorithm is that a node has a logical clock which provides a time base for all activities on a node [51], and which is derived from the hardware clock on the node. The algorithm executed can be viewed as a clock process invoked at the end of every resynchronization interval. This process is responsible for periodically reading the clock values at other nodes and then adjusting the corresponding local clock value. The algorithm must satisfy the following two conditions:

1. Agreement: The offset between all non-faulty clocks in the system is bounded.

2. Accuracy: The logical clock keeps up with the real time.

The diverse algorithms differ in the way these conditions are specified and in the way the clock processes read the clock values at the other nodes. They can be divided into two categories:

1. Deterministic: These guarantee an upper bound on the closeness of the synchronization attainable, assuming an upper bound on transmission delays [52–55].

2. Probabilistic: These do not make any assumptions about the uncertainty, and provide a smaller *expected* upper bound on the closeness of synchronization, but do not guarantee such a bound with probability one [56, 57].

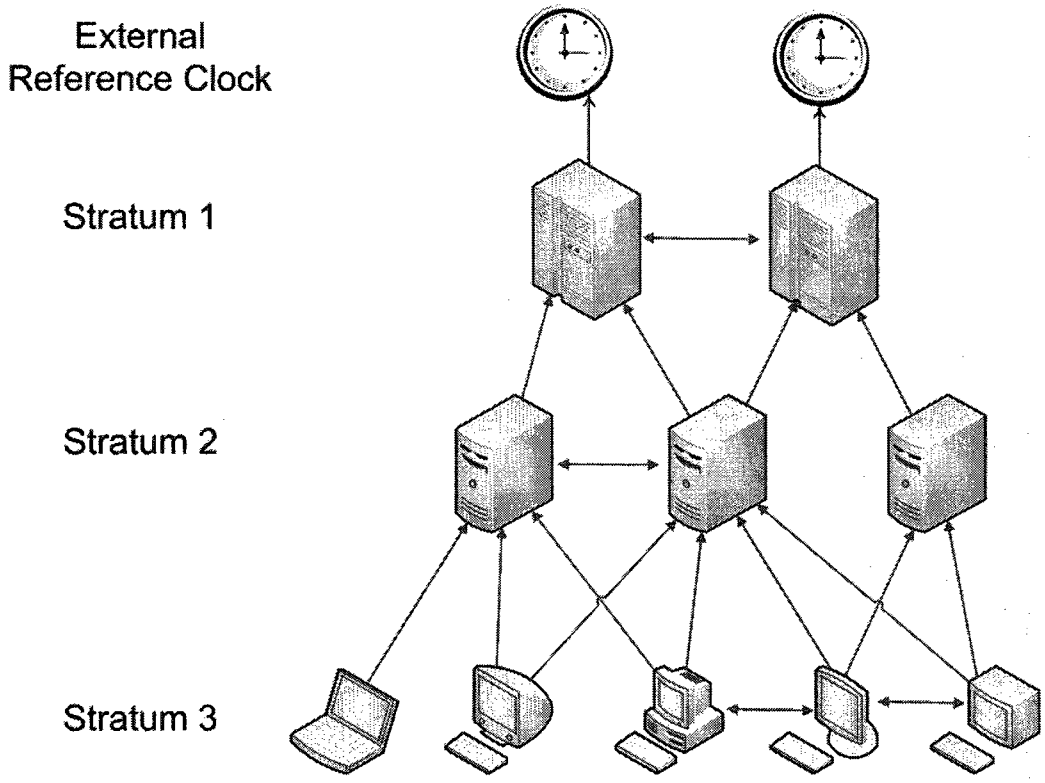


Figure 2.1: Relationship between the various levels of NTP.

One important algorithm which is widely used is the Network Time Protocol (NTP) described in [12]. As can be seen in Figure 2.1, in NTP each group of processes served by the synchronization protocol is organized in a hierarchical structure. The primary servers are located at the root level or stratum 1, and are synchronized to accurate external clocks. Nodes at stratum 2 are synchronized with the primary servers at stratum 1, nodes at stratum 3 are synchronized with nodes at stratum 2, and so on. To withstand possible failures in nodes or network links, nodes at the different strata can obtain time readings from several nodes, resulting in a redundant topology. To achieve synchronization, nodes exchange messages which include the

latest three timestamps that, together with the time at which the message is received, allow estimating round-trip delays and clock offsets. These estimates are then filtered to reduce timing noise, and a peer-selection algorithm is used to determine which subset is the most accurate and reliable. The resulting offsets of this subset are combined using a weighted-average basis, and processed by a phase-lock loop to produce a phase-correction term used to control the local clock. Levels close to the root of the subnet have better accuracy and precision relative to the external standard.

2.4 Ad Hoc Networks

An ad hoc network is a network of mobile wireless computing devices which have a possibly dynamical topology, there being no infrastructure. An algorithm for clock synchronization suitable for ad hoc networks is proposed in [13]. The basic idea of this algorithm is not to synchronize the local clocks of the devices but to generate time stamps using unsynchronized local clocks that when passed between devices will be transformed to the local time of the receiving device. Since these transformations cannot be done with high precision due to the unpredictability of the computer clocks, the algorithm estimates a lower and upper bound for the real time passed from the generation of a timestamp to its arrival in the destination, transforms such bounds to the time of the receiver, and subtracts the resulting values from the time of arrival in the destination node. This results in an interval specifying lower and upper bounds for the time stamp relative to the local time of the receiving node.

The accuracy obtained is in the order of milliseconds for an implementation over a 100 Mbps Ethernet network. A time translation control time protocol is also developed in [11] which is suitable for control applications over networks. It is also shown that it is impossible to synchronize under asymmetric delays, i.e., when communica-

tion delays in the two directions are different.

2.5 Wireless Sensor Networks

For wireless sensor networks which share the limitations of an ad hoc network, along with a need, generally, for greater energy efficiency, a few algorithms have been proposed. Before describing the more important algorithms in this area, we review what are the sources of error in the clock synchronization process [14, 58], and describe the most common architecture used to implement the algorithms for sensor networks, Berkeley motes [59].

2.5.1 Sources of error in the clock synchronization process

First we define some terminology.

Send time. This is the time spent at the sender to pass the message from the application to the network interface.

Access time. This is the delay incurred waiting for access to transmit on the channel, which in turn is dependent on the MAC protocol being used.

Propagation time. This is the time in transit from the sender to the receiver, once the message has left the sender. This time is very small on a local network if the sender and receiver share access to the same physical media.

Receive time. This is the time needed to process the message once it has arrived at the receiver's network interface.

These quantities affect the latencies of communications between nodes, and are shown graphically in Figure 2.2. Figure 2.3 shows the difference between timestamping at the application layer and the MAC layer and how some of sources of error can be avoided.

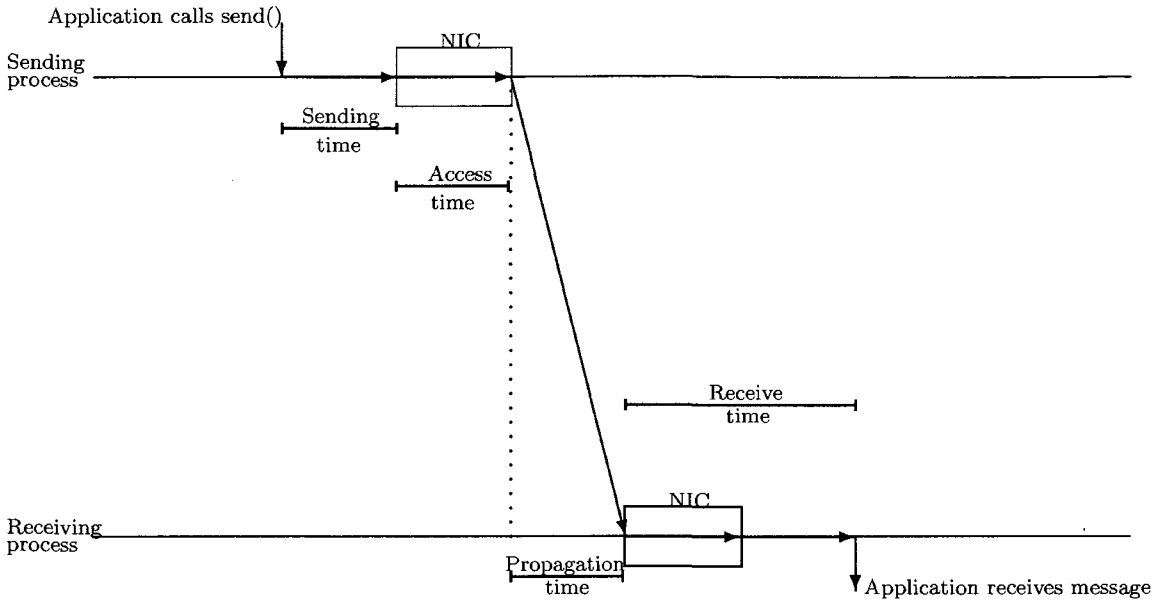


Figure 2.2: Illustration of the sources of error during clock synchronization.

2.5.2 Berkeley motes and TinyOS

A mote is an autonomous sensor node which provides a combination of sensing, communication and computation, in a complete architecture which has been developed at the University of California, Berkeley [59].

Berkeley motes run the TinyOS Operating System, which is an open-source operating system designed for wireless embedded sensor networks. Its component-based architecture enables rapid implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks [60]. Applications are written in nesC, which is a new language for programming structured component-based applications that has a C-like syntax and supports the TinyOS concurrency model [61].

Over the years, motes architecture has been improved in terms of processing power, memory and communication capabilities. We describe some of the features of three different types of motes to illustrate these improvements (for more details, see [62, 63]).

A MICA mote [64] uses an Atmel ATmega 103 or ATmega 128L processor running

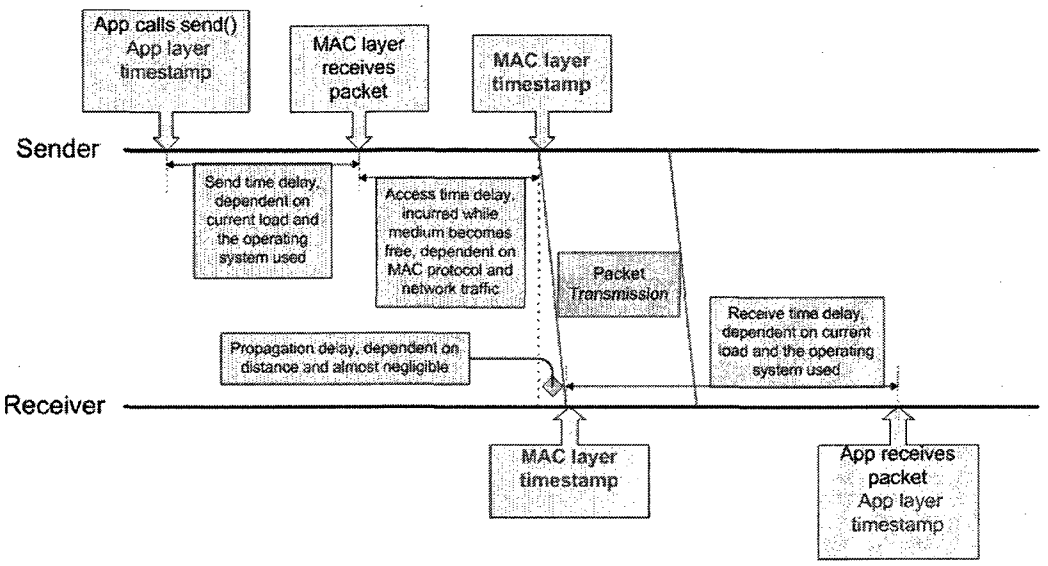


Figure 2.3: Illustration of two places where the timestamping can be performed.

at 4 MHz. It has 128 KBytes of onboard flash memory to store the mote's program and 4 KBytes of RAM.

It comes with 512 KBytes of flash memory to hold data, and also has a 10-bit A/D converter so that sensor data can be digitized. Sensors available include temperature, acceleration, light, sound and magnetic. Finally, the radio module consists of an RF Monolithics TR1000 transceiver that can operate at communication rates up to 40 kbps. The complete system is designed to operate off an inexpensive pair of AA batteries that produce between 2.0 and 3.2 V.

A MICA2 mote [8] has similar features as the MICA, but uses a different radio, the Chipcon CC1000, which allows a better range and better noise immunity. It also allows us to program the frequency at which to transmit the data, and has support for wireless remote reprogramming which is very helpful when there is a large number of motes to program. The processor has a 7.3728MHz crystal to support higher UART baud rates.

An IMote2 mote [65], which is a newer generation of mote, uses an Intel PXA271 XScale processor at 13 MHz (scalable up to 416 MHz). It has 256 KB of SRAM and

32 MB of SDRAM. It uses a CC2420 IEEE 802.15.4 radio module which supports a 250 kbps rate with 16 channels in the 2.4 GHz band.

The software side of the architecture, that is, TinyOS, has also been improved over the years [60]. In 2001, MICA motes were developed and programmed with version 0.6 which used a mix of C and Perl scripts. In September 2002, version 1.0 appeared, which was implemented in nesC. In November 2006, version 2.0 was released [66]; which is a complete rewrite of the entire operating system looking to support greater platform flexibility by means of a newer and more powerful version of the nesC language and the introduction of a three layer Hardware Abstraction Architecture [67]. The most recent version is 2.1.

2.5.3 Reference broadcast synchronization (RBS)

This scheme, described in [14], uses broadcast communications to allow the receivers of the synchronization message (called *beacon*) to synchronize with one another. The receivers record their local time when receiving the beacon, and then they exchange their recorded times. With this exchange, they find their clocks' difference. By doing this, two of the sources of error mentioned in Section 2.5.1 are removed (sending time and access time), and since the propagation time is ignored (considered being negligible in the broadcast medium in which this scheme is used), the only possible error is reduced to the receiver's side (receive time), and can be generally kept very small by timestamping the reception at the lowest possible level. The receiver error was characterized by doing tests on a wireless sensor network testbed in which a node broadcasted packets and the phase offsets of five receivers were recorded. The distribution of such errors appeared Gaussian with parameters $\mu = 0, \sigma = 11.1 \mu sec.$

In order to increase the precision of the synchronization, more than one beacon is sent:

1. m reference packets (beacons) are broadcasted.

2. Each of the n receivers records the time at which the beacon was received, according to its local clock.

3. The receivers exchange their recorded times.

4. Each receiver i can compute its offset with respect to any other receiver j as the average of the phase offsets implied by each beacon received by both nodes i and j,

$$\forall i \in n, j \in n : \text{Offset}[i,j] := \frac{1}{m}\sum_{k=1}^{m}(T_{j,k} - T_{i,k}).$$

To account for the clock skew, a least-squares linear regression is performed on the phase offsets. The protocol was implemented on a Berkeley motes testbed and it was able to keep the synchronization error between two nodes at 7.4 μs after a 60 second interval.

It should be noted that the protocol code ran on iPAQs that have more stable oscillators and a higher resolution (the motes were only used to provide the wireless communication). A multihop extension is also proposed to synchronize at least two groups of nodes, but it relies on effective clustering of the nodes around the broadcast nodes.

To reduce communications, a scheme called post-facto synchronization is proposed where instead of keeping the time-synchronization process always on, a synchronization is performed only after an event of interest happens, in order to estimate the phase shift at the previous time by extrapolating backwards. This however can result in delayed reaction to events, which can potentially affect the stability of sensor-actuator networks.

2.5.4 Tiny-sync and mini-sync

The work described in [15] assumes that the clock drift at a node is linear and of the form

$$t_i(t) = a_i t + b_i,$$

where t_i is the local clock in node i, a_i and b_i are its drift and offset parameters, and t is the real time. If node 1 wants to determine what are its relative drift and offset with respect to node 2 (a_{12} and b_{12} respectively), an exchange of messages would take place in the following way (shown graphically in Figure 2.4):

1. Node 1 sends a message to node 2, which is timestamped right before it is sent at t_o.

2. Node 2 timestamps the reception with t_b, and returns a reply to node 1 which includes t_b.

3. Node 1 timestamps the reply with t_r.

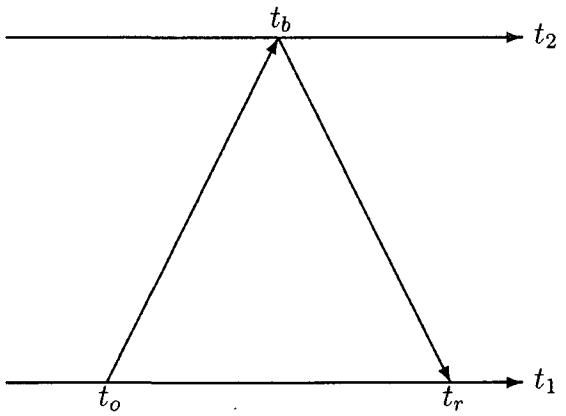


Figure 2.4: Exchange of messages to determine relative drift and offset between two nodes.

The three timestamps (t_o, t_b, t_r) form a data-point which limits the values of parameters a_{12} and b_{12}. Since $t_0 \rightarrow t_b$ and $t_b \rightarrow t_r$, the following inequalities should hold for the data points:

$$t_o(t) < a_{12} t_b(t) + b_{12} \,,$$

$$t_r(t) > a_{12} t_b(t) + b_{12} \,.$$

After the acquisition of a few (at least two) data-points, an estimation of the parameters can be performed by solving the linear programming problem given by the inequalities formed by the data-points. As new data-points arrive, the number of data-points to keep can be reduced to only two by determining which ones result in the best estimates (tiny-sync method). Alternatively, up to 40 points can be used with a different way to eliminate old data-points (mini-sync method). The tiny-sync method results in a suboptimal solution.

To synchronize a multihop network, it is assumed that the sensor network is organized as a tree hierarchy with sensor nodes in the lowest layer, one (or more) root node(s), and possibly several layers of intermediate nodes. It is also assumed that data is fused at the intermediate nodes. So, instead of synchronizing the entire network to one unique clock, all nodes reporting to the same intermediate node should synchronize with such intermediate node using the exchange of messages described. Therefore, nodes in layer i synchronize with nodes in layer $i-1$ and so on. The performance of the algorithm was tested on an 802.11b multihop ad hoc network, achieving an accuracy of 3 ms over a single hop.

2.5.5 Timing-sync protocol for sensor networks (TPSN)

In [16], a protocol, called TPSN, is proposed to achieve network-wide synchronization. In this protocol, the first step is to create a hierarchical topology in the network where

every node is assigned a level in the hierarchical structure. To create this hierarchical topology, the root (which is assigned level 0) initiates a level discovery phase when the network deploys, by broadcasting a level_discovery packet, which contains the identity and level of the sender. Once the neighboring nodes receive this packet, they assign themselves a level which is one greater than the level they have received and broadcast a new level_discovery packet. This process continues until every node is assigned a level. Once a node has assigned itself a level, then further level_discovery packets received are discarded.

Once the level discovery phase ends, a synchronization phase starts where a node belonging to a level i synchronizes to level i-1. In this latter phase, a two-way message exchange occurs, and the clock drift and propagation delay are calculated, with the node making the calculations and correcting its clock accordingly. The two-way message exchange to synchronize two nodes A and B works in the following way (see Figure 2.5):

1. Node A sends a message (called synchronization pulse) to B at time T1. This message contains the level number of A and the value of T1.

2. Node B receives the message at time T2, where $T2 = T1 + \delta + d$, δ and d being the clock drift between the two nodes, and the propagation delay, respectively.

3. Node B sends back an acknowledgment to node A at time T3, containing the level number of B and the values of T1, T2 and T3.

4. Node A receives the acknowledgment at T4.

Once the acknowledgment is received at node A, it can calculate the clock drift δ

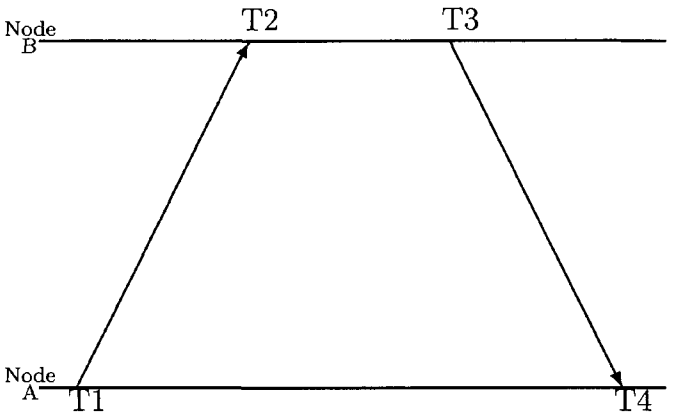


Figure 2.5: Two-way message exchange.

and propagation delay d as follows:

$$\delta = \frac{(T2 - T1) - (T4 - T3)}{2},$$

$$d = \frac{(T2 - T1) + (T4 - T3)}{2}.$$

This protocol was implemented on Berkeley MICA motes using a 4MHz crystal oscillator, after making modifications to the MAC layer so it could timestamp the messages at reception time. The accuracy obtained is around 17μs for the synchronization of two motes.

2.5.6 Flooding time synchronization protocol (FTSP)

In [17], Flooding Time Synchronization Protocol (FTSP) is proposed to achieve clock synchronization on a wireless sensor network. It uses MAC layer timestamping capabilities to eliminate several sources of error in the clock synchronization process: sending time, access time and receive time. In this protocol, it is assumed that a node, the root, which is elected after the exchange of messages in the network, is already

synchronized, and the other nodes synchronize with this node. The root broadcasts a message which includes the global time at the time of sending the message and root ID. The receiver gets a timestamp at the time of reception, and using the arrival and sending times, it estimates the clock offset and skew once a number of readings are available, using linear regression.

When a mote determines it is already synchronized, it also broadcasts packets which include the global time and root ID, creating in this fashion a clock synchronization hierarchy where the root is at Level 0, nodes within the broadcast range of the root at Level 1 and so on. Every node therefore estimates the global time by synchronizing its clock to the nodes one level higher than itself.

This protocol has been implemented in Berkeley MICA2 motes. The average accuracy obtained for a 60-node network is 16 μs.

2.5.7 Lightweight tree-based synchronization (LTS)

[18] proposes a tree-based synchronization method for sensor networks that requires the construction of a spanning tree in order to generate the pair-wise synchronizations required to synchronize the whole network. It also proposes another method to synchronize a multihop network without requiring the construction of a tree, but assumes the use of a multi-channel MAC. Both versions were simulated but not implemented, and are aimed at minimizing the complexity of the synchronization and not in maximizing accuracy.

2.5.8 Adaptive clock synchronization

In [19], RBS is extended to provide an adaptive and probabilistic clock synchronization allowing a trade-off between the accuracy achieved and the resources used by the protocol. It has not been implemented.

2.5.9 Pairwise broadcast synchronization (PBS)

In [20], a new approach for clock synchronization is proposed. It uses the broadcast nature of the wireless channel, and allows a node to synchronize by overhearing synchronization packets exchanged in a similar fashion as TPSN among two neighboring nodes, without the need to send extra messages. This method reduces the number of messages required for synchronization but requires that all nodes hear each other. A multi-cluster extension has been proposed in [21], which requires the construction of a hierarchical tree and the use of groupwise pair selection algorithm to achieve global synchronization.

2.5.10 Gradient time synchronization protocol (GTSP)

The Gradient Time Synchronization Protocol (GTSP) is proposed in [22]. The idea of this protocol is to provide a precise clock synchronization between neighboring nodes and allow a more loose synchronization between nodes separated by multiple hops. The algorithm does not require the construction of a hierarchical topology, or a reference node, but in order for it to converge, the network needs to be strongly connected. It has been implemented on a small network of 20 Berkeley MICA2 motes, obtaining an average error of 4 μs for direct neighbors, and a network-wide average error of 14 μs.

2.5.11 Average time sync (ATS)

[23] proposes the Average Time Sync (ATS) to synchronize a multihop wireless sensor network. The protocol is formulated as a consensus problem, and achieves global clock synchronization to a virtual reference clock in a distributed manner. The protocol works by performing three tasks in each of the nodes: relative skew estimation, skew compensation and offset compensation. These tasks require only local information

exchanges between neighboring nodes, each of the nodes maintains its own estimate of the virtual reference clock, which is updated by averaging it with respect to the estimate that the neighboring nodes have. It has been implemented on Tmote Sky nodes [68].

As can be observed from the description of the protocols for clock synchronization in wireless sensor networks, with the exception of the last two ([22,23]), either they rely on the construction of a hierarchical tree structure ([15–18]), which generates a lot of overhead and does not support dynamic topology changes very well, or they rely on all the nodes sharing the media with a beaconing node ([14,19,20]), which makes them unsuitable for multihop networks.

The protocol described in Chapter 4 provides an efficient clock synchronization algorithm for wireless multihop networks without requiring the construction of a hierarchical structure. It is fully distributed and has been implemented on a Berkeley motes testbed with very good results.

CHAPTER 3

BILATERAL CLOCK SYNCHRONIZATION ALGORITHM

This chapter presents the bilateral synchronization part of our new clock synchronization algorithm. The system model is described in Section 3.1. The method to achieve a pair-wise synchronization between neighboring nodes is explained in Section 3.2.

3.1 System Model

It is assumed in this work that the clock drift in a node follows the linear equation: $T_i = \alpha_i t + O_i$, just as in [11,15,17], where T_i is the local clock, α_i and O_i are the drift parameters that express the relative speed of the clock and the offset respectively, and t is the real time. Nodes' clocks drift at different rates. Neighboring nodes exchange timestamps to estimate the best-fit offset line between them by using a recursive least squares (RLS) estimation approach. Further details are provided in Section 3.2.

Also assumed is the fact that all the nodes are aware of the set of neighbor nodes with which they can directly communicate, and that the communication links between the nodes are bidirectional.

3.2 Pair-wise Clock Synchronization between Neighbors

In this section we very briefly describe how bilateral synchronization between two neighboring nodes is performed. This is not really novel, since it is just a variant of

linear regression.

Suppose we have two nodes i and j which can communicate directly, and want to determine their offset O_{ij} and skew α_{ij} by exchanging packets. By the offset $O_{ij}(t)$ we will mean the difference in the two clocks $T_j(t) - T_i(t)$. By the skew α_{ij} we will mean the ratio of the speeds $\frac{\alpha_j}{\alpha_i}$.

At time $X(t_k)$, node i sends a packet $p(t_k)$ that includes this timestamp, its latest estimate of its skew with respect to j, $\hat{\alpha}_{ji}(t_k)$, and its latest estimate of the time difference between received and transmitted timestamps for packets from j to i, $\Gamma_{ji}(t_k)$. Packet $p(t_k)$ is received at time $U(t_k)$ by node j. This is done frequently as can shown in Figure 3.1. With these packets, node j can compute the value $\hat{\alpha}_{ij}(t_k)$, which is the estimate of the skew of j with respect to i at real time t_k.

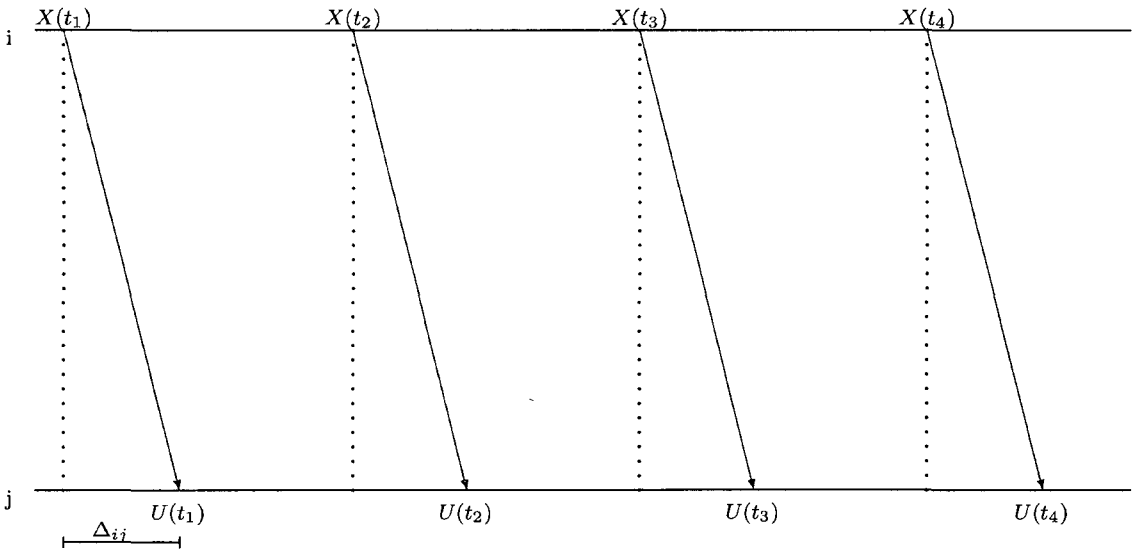


Figure 3.1: Frequent transmissions of estimates to neighbor node.

The value $\hat{\alpha}_{ij}(t_k)$ is estimated using Recursive Least Squares (RLS) [69] to solve the following problem:

$$\hat{\alpha}_{ij}(t_k) = \min_{\alpha} \sum_{l=-\infty}^{k-1} \lambda^{k-1-l} \left[U(t_{l+1}) - U(t_l) - \alpha \left[X(t_{l+1}) - X(t_l) \right] \right]^2, \quad (3.1)$$

where $\lambda \in (0,1)$ is the forgetting or weighting factor that reduces the influence of old data.

The RLS algorithm is widely used in adaptive filtering, system identification and adaptive control. It also allows us to estimate $\hat{\alpha}$ with minimal storage requirements since only the previous and current values of $U(t)$ and $X(t)$ are required at any given time.

To develop the algorithm, we rewrite equation (3.1) as follows:

$$\hat{\Delta}_k = \min_{\Delta} \sum_{l=-\infty}^{k-1} \lambda^{k-1-l} \left[\Upsilon_l - \phi_l^T \Delta \right]^2, \tag{3.2}$$

where

$$\begin{aligned} \Upsilon_l &= |U(t_{l+1}) - U(t_l)|, \\ \phi_l^T &= |X(t_{l+1}) - X(t_l)|, \\ \Delta &= |\alpha| . \end{aligned}$$

We differentiate (3.2) with respect to Δ and set it to zero, i.e.,

$$\frac{d}{d\Delta} \left[\sum_{l=-\infty}^{k-1} \lambda^{k-1-l} \left[\Upsilon_l - \phi_l^T \Delta \right]^2 \right] = 0,$$

and get the following:

$$\sum_{l=-\infty}^{k-1} \lambda^{k-1-l} \phi_l \left[\Upsilon_l - \phi_l^T \widehat{\Delta}_k \right] = 0,$$

where $\widehat{\Delta}_k$ is the least squares estimate. Hence,

$$\sum_{l=-\infty}^{k-1} \lambda^{k-1-l} \phi_l \Upsilon_l \;=\; \left[\sum_{l=-\infty}^{k-1} \lambda^{k-1-l}\, \phi_l \phi_l^T \right] \widehat{\Delta}_k. \tag{3.3}$$

Let us define

$$R_{k-1} := \sum_{l=-\infty}^{k-1} \lambda^{k-1-l}\, \phi_l \phi_l^T \,. \tag{3.4}$$

Then we can see that

$$R_k = \lambda R_{k-1} + \phi_k \phi_k^T \,. \tag{3.5}$$

Substituting (3.4) in (3.3) we get

$$R_{k-1} \widehat{\Delta}_k = \sum_{l=-\infty}^{k-1} \lambda^{k-1-l}\, \phi_l \Upsilon_l \,.$$

Similarly, we obtain

$$\begin{aligned} R_k \widehat{\Delta}_{k+1} &= \sum_{l=-\infty}^{k} \lambda^{k-l}\, \phi_l \Upsilon_l \\ &= \sum_{l=-\infty}^{k-1} \lambda^{k-1-l}\, \phi_l \Upsilon_l + \phi_k \Upsilon_k \\ &= R_{k-1} \widehat{\Delta}_k + \phi_k \Upsilon_k \\ &= \left(R_k - \phi_k \phi_k^T \right) \widehat{\Delta}_k + \phi_k \Upsilon_k \,. \end{aligned}$$

Hence, multiplying by R_k^{-1}, assumed to exist, we obtain

$$\begin{aligned} \widehat{\Delta}_{k+1} &= \left(I - R_k^{-1} \phi_k \phi_k^T \right) \widehat{\Delta}_k + R_k^{-1} \phi_k \Upsilon_k \\ &= \widehat{\Delta}_k + R_k^{-1} \phi_k \left(\Upsilon_k - \phi_k^T \widehat{\Delta}_k \right) \,. \end{aligned} \tag{3.6}$$

Now, we need to find a way to calculate R_k in a recursive fashion using only the information available at time k. For this, we use the Matrix Inversion Formula [69] that says the following:

> If A and B are M x M invertible matrices, D is a N x N matrix, and C is a M x N matrix, which are related by
>
> $$A = B^{-1} + CD^{-1}C^T,$$
>
> then
>
> $$A^{-1} = B - BC\left(D + C^T BC\right)^{-1} C^T B,$$
>
> whenever the inverses exist.

We apply this formula to (3.5) using

$$\begin{aligned} D &= 1, \\ C &= \phi_k, \\ B^{-1} &= \lambda R_{k-1}, \end{aligned}$$

and

$$A = R_k,$$

and get the following

$$\begin{aligned} R_k^{-1} &= (\lambda R_{k-1})^{-1} - (\lambda R_{k-1})^{-1} \phi_k \left(1 + \phi_k^T (\lambda R_{k-1})^{-1} \phi_k\right)^{-1} \phi_k^T (\lambda R_{k-1})^{-1} \\ &= \frac{R_{k-1}^{-1}}{\lambda} - \frac{R_{k-1}^{-1} \phi_k \phi_k^T R_{k-1}^{-1}}{\lambda\left(\lambda + \phi_k^T R_{k-1}^{-1} \phi_k\right)}. \end{aligned}$$

Now, from (3.6) we see that we need to compute $R_k^{-1}\phi_k$, so

$$\begin{aligned} R_k^{-1}\phi_k &= \frac{R_k^{-1}\phi_k}{\lambda} - \frac{R_k^{-1}\phi_k\phi_k^T R_{k-1}^{-1}\phi_k}{\lambda\left(\lambda + \phi_k^T R_{k-1}^{-1}\phi_k\right)} \\ &= \frac{R_k^{-1}\phi_k\left(\lambda + \phi_k^T R_{k-1}^{-1}\phi_k\right) - R_{k-1}^{-1}\phi_k\phi_k^T R_{k-1}^{-1}\phi_k}{\lambda\left(\lambda + \phi_k^T R_{k-1}^{-1}\phi_k\right)} \\ &= \frac{R_{k-1}^{-1}\phi_k}{\lambda + \phi_k^T R_{k-1}^{-1}\phi_k}. \end{aligned}$$

Therefore, we can now compute $\widehat{\Delta}_{k+1}$ using values that are all known at time k with the following recursion:

$$\widehat{\Delta}_{k+1} = \widehat{\Delta}_k + \frac{R_{k-1}^{-1}\phi_k}{\lambda+\phi_k^T R_{k-1}^{-1}\phi_k}\left(\Upsilon_k - \phi_k^T\widehat{\Delta}_k\right).$$

Once the skew $\hat{\alpha}_{ij}(t_k)$ is computed, we can estimate the time at which the current packet should be received, using a window of N values of $U(t_k)$ and $X(t_k)$, by doing the following computation:

$$\hat{U}(t_k) = \frac{1}{N}\sum_{l=k-N}^{k-1}\left[U(t_l) + \bar{\alpha}_{ij}(t_k)\left(X(t_k) - X(t_l)\right)\right], \tag{3.7}$$

where

$$\bar{\alpha}_{ij}(t_k) := \sqrt{\frac{\hat{\alpha}_{ij}(t_k)}{\hat{\alpha}_{ji}(t_k)}}. \tag{3.8}$$

Then, we can determine the difference between the received and transmitted timestamps at nodes j and i, Γ_{ij}. This is the sum of the offset of node j with respect to i, O_{ij}, and the transmission delay of the packet, Δ_{ij}, at time t_k. It is estimated by subtracting $X(t_k)$ from $\hat{U}(t_k)$:

$$\Gamma_{ij}(t_k) = O_{ij}(t_k) + \Delta_{ij} = \hat{U}(t_k) - X(t_k). \tag{3.9}$$

However, we need to first determine the value of Δ_{ij} so that we can estimate the value of $O_{ij}(t_k)$. In order to achieve this, packets must be sent back from node j to node i and should include the following values: $\hat{\alpha}_{ij}$, $\Gamma_{ij}(S)$ and the time S at which the packet was sent. $\Gamma_{ij}(S)$ is then computed as follows;

$$\Gamma_{ij}(S) = \Gamma_{ij}(t_k) + \left(S - \hat{U}(t_k)\right) \frac{\bar{\alpha}_{ij}(t_k) - 1}{\bar{\alpha}_{ij}(t_k)}, \tag{3.10}$$

to account for the change in offset between $\hat{U}(t_k)$ and S due to skew.

Node i executes the same procedure as node j. That is, it also estimates its skew with respect to j, $\hat{\alpha}_{ji}(t_k)$, the time at which the packet should have been received, and its difference in time with respect to j, $\Gamma_{ji}(t_k)$.

Now, node i can estimate the value of the transmission delay as

$$\hat{\Delta}_{ji}(t_k) = \frac{\Gamma_{ji}(t_k) + \Gamma_{ij}(t_k)}{2}, \tag{3.11}$$

and then its offset with respect to j as

$$\hat{O}_{ji}(t_k) = \Gamma_{ji}(t_k) - \hat{\Delta}_{ji}(t_k). \tag{3.12}$$

Now, on the next packet, $p(t_{k+1})$, to be sent from node i to j, the values included in it would be its most recently calculated skew, $\hat{\alpha}_{ji}(t_k)$, its most recently calculated transmission delay, $\hat{\Delta}_{ji}(t_k)$, and the estimated offset at time t_{k+1}, which is given by

$$\hat{O}_{ji}(t_{k+1}) = \hat{O}_{ji}(t_k) + \left(t_{k+1} - \hat{U}(S)\right) \frac{\bar{\alpha}_{ji}(t_k) - 1}{\bar{\alpha}_{ji}(t_k)}. \tag{3.13}$$

This process is then repeated.

By this method we can obtain estimates of offsets and skews at given times between two neighboring nodes.

3.3 Simulation Results

Before implementing the algorithm discussed, simulations were performed to verify if the behavior of the algorithm was as expected.

First, we simulated the pair-wise synchronization procedure with different values of λ and N (forgetting factor for the RLS algorithm and window size to estimate $U(t)$, respectively). We found through simulation with various values of λ and N that values of $\lambda = 0.999$ and $N = 10$ provide the best estimation accuracy without requiring too much memory space.

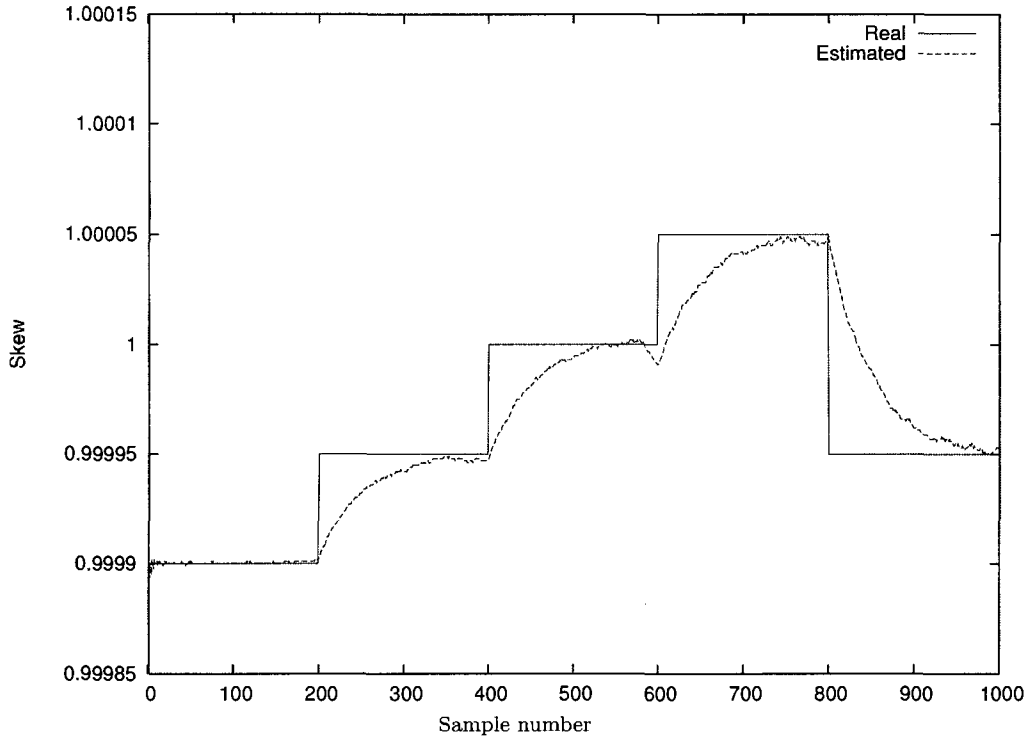


Figure 3.2: Skew estimation results obtained in simulation.

Then, the skew of one of the nodes simulated was modified frequently, within the ranges ($\pm 100ppm$) of the physical oscillator [70] found on the experimental testbed to be used, during the course of the simulations in order to determine if the calculations of skew and offset adapt to the changes. We can see in Figure 3.2 that the skew

estimate indeed adapts to the real skew as it changes. Also, in Figure 3.3 we see that the difference between the real and estimated offsets between the two nodes is below 2 μs.

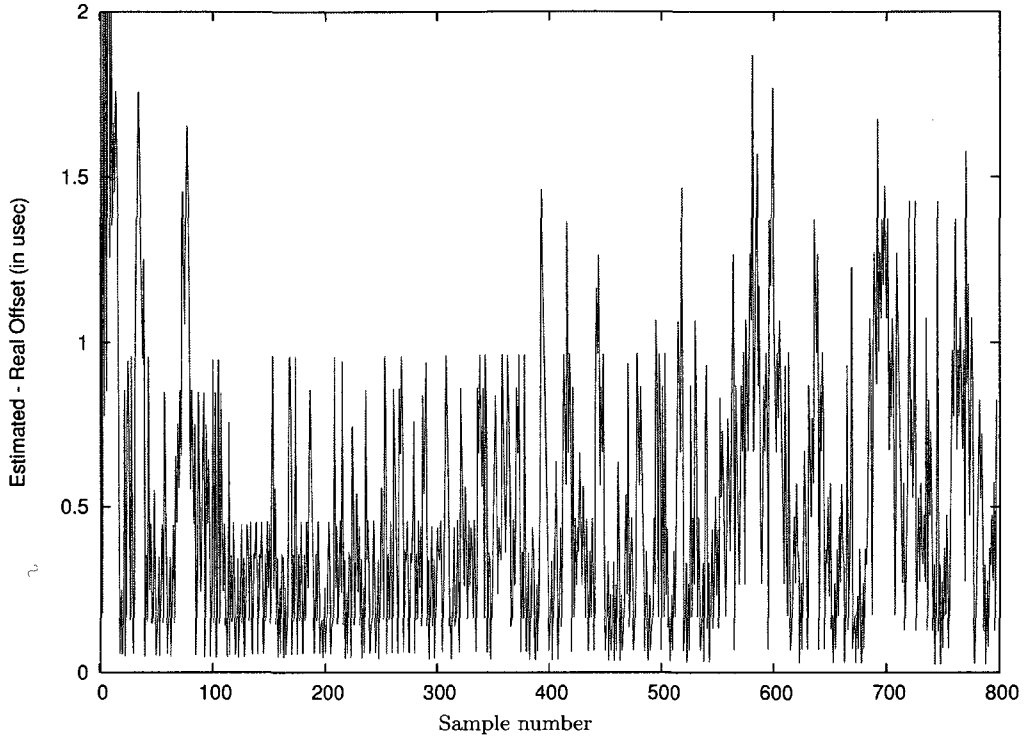


Figure 3.3: Offset estimation results obtained in simulation.

3.4 Implementation on Berkeley Motes

For the implementation of the algorithm described in this chapter, as well as the one that will be presented in the next chapter, we used a testbed of MICA and MICA2 motes.

Figure 3.4, which shows the communication stack of TinyOS on the Berkeley motes, provides a better understanding of the sources of error in the clock synchronization process mentioned in Section 2.5.1 and how can we prevent some of them in TinyOS.

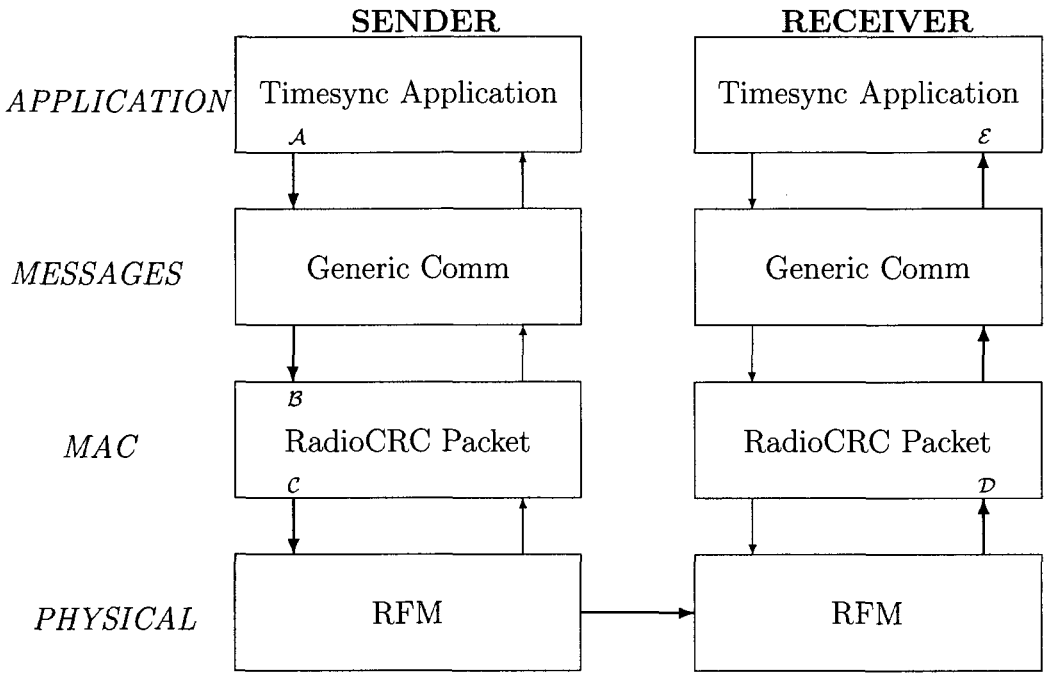


Figure 3.4: Communication stack on TinyOS 1.x.

- The message to be sent is constructed at the application layer and is then passed to the lower layers for its transmission. The time that it takes to construct the message and pass it down to the MAC layer is the *send time*. In Figure 3.4, it would be time period starting at the point the Clock Synchronization application calls sendMsg() (denoted by the symbol $\mathcal{A}$), and ending at the reception of the message at the RadioCRCPacket component (denoted by the symbol $\mathcal{B}$).

- Once in the MAC layer, some time must elapse before the message is actually transmitted over the medium, since it has to wait until it has been determined that the medium is idle. This is the *access time*. In Figure 3.4, it would be the time period starting at the reception of the message at the RadioCRCPacket component (denoted by the symbol $\mathcal{B}$), and ending at the time the first byte is sent for transmission to the Radio Module (denoted by the symbol $\mathcal{C}$).

- Once the bits have been received at the receiver, the message is reconstructed and passed up to the application layer. The time taken to do all this is the *receive time*. In Figure 3.4 it would be time period starting at the reception of the first byte at the RadioCRCPacket component from the Radio Module (denoted by the symbol $\mathcal{D}$), and ending at the time the message is received at the Clock Synchronization application (denoted by the symbol $\mathcal{E}$).

Therefore, in order to reduce the sources of error in our implementation, we have modified the MAC layer of TinyOS to allow the timestamping of a packet at the time the first byte is sent, instead of doing it at the application layer. We also record the time at which the first byte is received by the MAC layer at the receiver side for later use by the application layer. In this way, we attempt to mitigate the send time, access time and receive time as sources of error, leaving only the propagation time, which is negligible in the case of the broadcast medium used in the Berkeley motes.

3.4.1 Setup

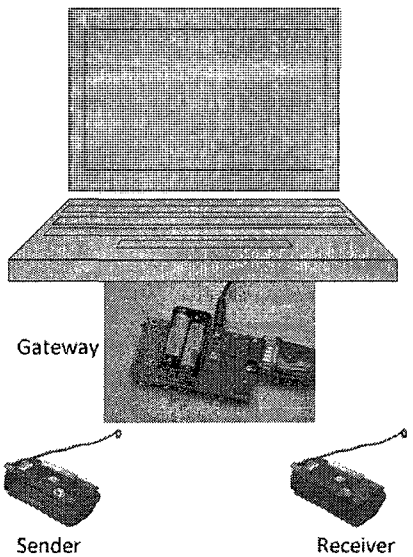


Figure 3.5: Implementation setup for two nodes.

For this first stage of testing, three motes were used for the implementation, the

sender and receiver as described before, and a third mote connected to a PC through the serial port acting as a data collector or gateway. This latter mote continuously senses the medium and communicates the data being transmitted to the PC. The sender and receiver exchange a clock synchronization message every 30 seconds. This setup is illustrated in Figure 3.5.

In order to get better accuracy, the motes' clocks were modified to generate a finer granularity clock, which is triggered by a crystal oscillator. The frequency of such crystals was modified in the MICA motes to 500 KHz. For this, the clock timer had to be changed since the original timer (Timer 0) uses an 8-bit register to maintain the clock counter. This would overflow at this frequency, and so we use instead Timer 1 which has a 16-bit clock counter. These changes were problematic since Timer 1 is already used for other purposes by TinyOS, and the conflicts raised had to be solved in order for TinyOS and our application to work correctly. For MICA2 motes, a component to use a higher frequency of 921.6 KHz was made available by Maroti et al. [71].

Although MICA motes were initially used for the implementation of the pair-wise synchronization, due to the lack of wireless reprogramming and support in newer versions of TinyOS, the subsequent experimentations were only performed on MICA2 motes.

3.5 Evaluation

In order to determine the accuracy of this protocol, the gateway mote periodically requests the time each mote has estimated. Both motes reply to this query, one with the time it has estimated at the other node, and the other node with its current time. The difference between the times replied is the accuracy or error of the protocol. By using this method to compare the times in both motes, we remove the send time,

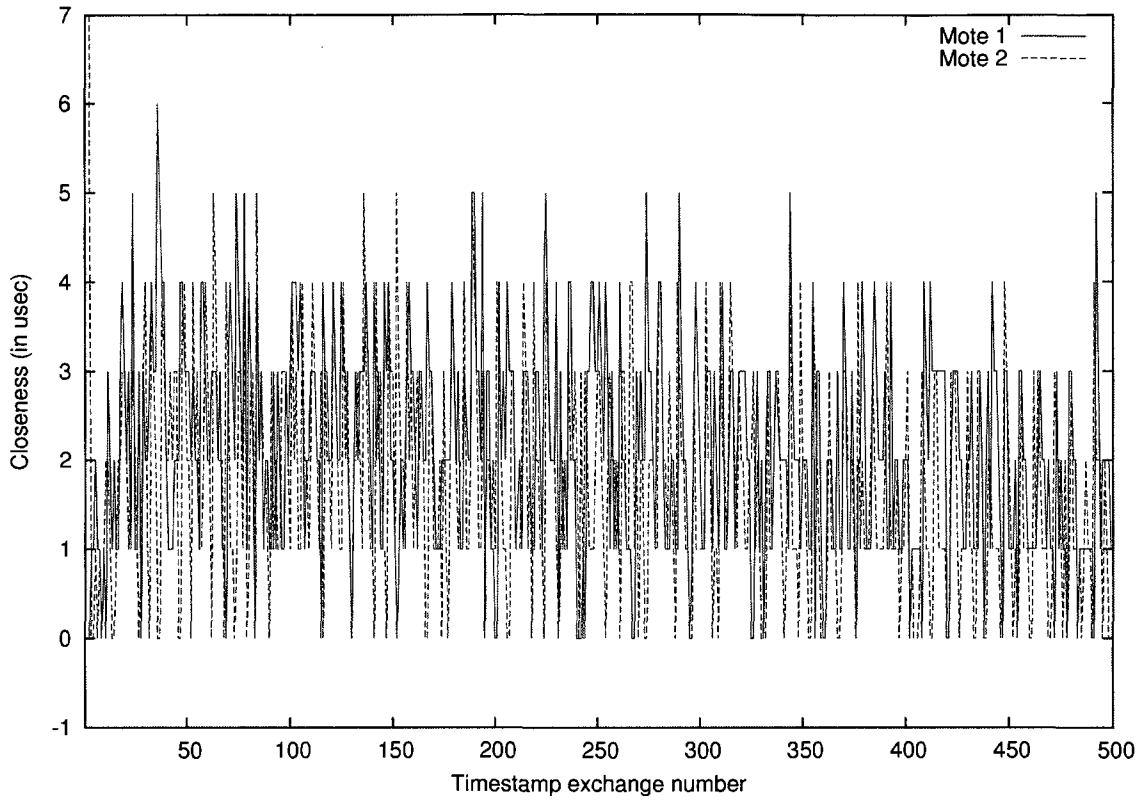


Figure 3.6: Accuracy of estimated to real time between 2 neighboring nodes.

access time, and propagation time as sources of error (see Figure 2.5.1).

With the implementation on MICA2 motes, we obtained an average accuracy of below $2\mu s$ (with a worst-case accuracy of $6\mu s$) as can be seen on Figure 3.6. This is better than the accuracy obtained in the related works mentioned in Section 2.5, with the exception of [17]. It has similar results as ours since we use the same timestamping technique to eliminate most of the sources of error that appear when we try to synchronize nodes in a network through message exchanges [58].

Now that each node is able to estimate the time at each of its neighbors, we want all nodes to be able to agree on a common time. In the next chapter we address this problem.

CHAPTER 4

MULTIHOP CLOCK SYNCHRONIZATION

We now turn to clock synchronization for a network of clocks. We address the problem of how to globally combine bilateral estimates of offset and skew into estimates of time with respect to the reference clock for each node in the network.

Just for simplicity of presentation, suppose that there are n nodes all having clocks running at exactly the same speed, except that they have different offsets. With node 1 chosen as the reference, let z_i denote the amount that node i's clock is ahead of node 1. So $z_1 = 0$. Thus, if $t_i(t)$ denotes the time at clock i at real time t, then

$$t_i(t) = t_1(t) + z_i$$

for all t. Let

$$x_{ij} := z_i - z_j$$

be the offset between nodes i and j.

Let N_i denote the set of neighbors of node i, and $|N_i|$ the number of such neighbors.

Assume that through experimentation as in Section 3.2, we have obtained estimates $\hat{x}_{ij}$ of $x_{ij} = (z_i - z_j)$ for $j \in N_i$ for all i. These estimates will not be exactly equal to the true values. We will also suppose that

$$\hat{x}_{ji} = -\hat{x}_{ij} \text{ for } j \in N_i \text{ for all } i,$$

so that nodes i and j have talked to each other in arriving at this estimate (using the algorithm described in Chapter 3).

The problem is this: We want to obtain estimates $v_i = \hat{z}_i$ of the offsets with respect to the reference node.

4.1 Formulation

Note that the true x_{ij}'s satisfy the global constraint

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_m i_1} = 0,$$

for every cycle

$$\ell = ((i_1, i_2), (i_2, i_3), \ldots, (i_m, i_1))$$

in the multihop network.

This is one example of a *global constraint* that needs to be satisfied by the offset between neighboring nodes. There are many such constraints because there are many cycles in the graph of the wireless network.

However, the estimates $\hat{x}_{ij}$ arrived at through only bilateral transactions described in Chapter 3 need not satisfy these constraints.

By enforcing the large number of such constraints, the estimates $\{\hat{x}_{ij}\}$ can be smoothed and improved. We call this procedure "spatial smoothing." In essence, it exploits a spatial law of large numbers through a reformulation of the problem to lead to better clock synchronization with respect to any chosen reference node.

Moreover, we would like to perform such constrained estimation over the ad hoc network with the nodes acting in a completely distributed asynchronous manner. We will now develop an algorithm where each node only needs to asynchronously broadcast a few numbers. Each node updates its numbers through an extremely simple formula based on each received broadcast. This then will be shown to lead to

an estimate of the offset with respect to any node which acts as a reference node.

Nodes will not need to know which is the reference node, or the topology of the network. No levels of hierarchies of nodes need to be constructed. In fact, our protocol exploits all edge offset estimates in the network, and not just those along a tree. In a large network it uses all the global information, and yet does so in a simple distributed asynchronous manner. Not all local broadcasts have to be received by any node either.

The manner of changing from one reference node to another, i.e., reference clock handoff, is also easy. The old reference node simply starts adapting, while the new one stops. The rest of the network automatically adapts to this change, and need not be explicitly informed of such reference handoffs.

We illustrate the method with a concrete example. For the network in Figure 4.1, there are five nodes $\{1, \ldots, 5\}$ and six arcs $\{(1,2), (2,3), (3,4), (1,4), (2,5), (3,5)\}$.

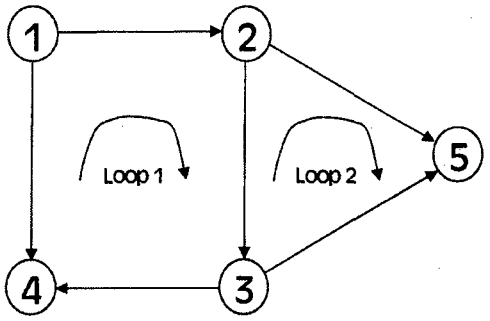


Figure 4.1: Example of a network.

As a convention we shall take each arc as going from a lower lexicographically numbered node to a higher one. Let A be the Node $\times$ Arc matrix, called the incidence

matrix:

$A =$

	(1,2)	(2,3)	(3,4)	(1,4)	(2,5)	(3,5)
1	+1	0	0	+1	0	0
2	-1	+1	0	0	+1	0
3	0	-1	+1	0	0	+1
4	0	0	-1	-1	0	0
5	0	0	0	0	-1	-1

where in the row corresponding to node i, we have an entry $+1$ for all arcs of the form $(i, *)$, an entry -1 for all arcs of the form $(*, i)$, and 0 otherwise.

Let v_i denote the estimate that we will make of z_i. Then the estimate of the offset between nodes i and j is $v_i - v_j$, which is the inner product, $\langle (ij)$-th column of A, vector $v \rangle$. Hence, written as a vector of estimates of arc offsets, it is $A^T v$.

Thus the problem formulation is:

$$\text{Min}_v \| A^T v - \hat{x} \|^2. \tag{4.1}$$

Note that this seeks the minimum norm approximation in the range space of A^T to the vector $\hat{x}$. The error of the approximation, by the Projection Theorem, is orthogonal to $R(A^T)$, the range space of A^T. That is, $(A^T v - \hat{x})$ is orthogonal to $R(A^T)$. However $R(A^T)^\perp = N(A)$, where $N(A)$ is the null space of A. Thus $(A^T v - \hat{x})$ is in the null space of A. Hence the solution of this optimization problem is

$$A(A^T v - \hat{x}) \;=\; 0,$$

or

$$AA^T v \;=\; A\hat{x}. \tag{4.2}$$

Now let us consider the i-th row of A. It corresponds to node i. It has a ± 1 for every incident arc. Now the j-th column of A^T similarly has ± 1 for every arc incident to j.

Hence $(AA^T)_{ii} = \#$ of neighbors of $i = |N_i|$. Also,

$$\begin{aligned}(AA^T)_{ij} &= -1 \text{ if } j \in N_i \\ &= 0 \text{ if } j \notin N_i.\end{aligned}$$

Thus the i-th entry of $AA^T v$ is

$$|N_i| v_i - \sum_{j \in N_i} v_j.$$

Also, the i-th entry of $A\hat{x}$ is the sum of terms of the form $+1$ times $\hat{x}_{i*}$, or (-1) times $\hat{x}_{*i}$, which in either case is $\hat{x}_{i*}$. Thus the i-th entry of $A\hat{x}$ is

$$\sum_{j \in N_i} \hat{x}_{ij}.$$

Thus (4.2) can be written as:

$$|N_i| v_i - \sum_{j \in N_i} v_j = \sum_{j \in N_i} \hat{x}_{ij} \text{ for all } i. \tag{4.3}$$

Note that this is deficient by at least one rank. So we set

$$v_1 := 0.$$

This corresponds to choosing node 1 as the reference node.

We will now consider the problem:

$$\text{Min}_v \sum_i \left(|N_i| v_i - \sum_{j \in N_i} v_j - \sum_{j \in N_i} \hat{x}_{ij} \right)^2. \tag{4.4}$$

We will solve the problem by coordinate descent since that will provide a fully distributed algorithm, in addition to being asynchronous. At the m-th iterate, let $v(m)$ be the estimate. We can take the initial iterate as

$$v(0) \;:=\; 0.$$

Also, since node 1 is the reference, we also have

$$v_1(m) \;:=\; 0 \quad \text{for all} \quad m \geq 0.$$

We will minimize over $v_i(k)$ perturbing it by δ_i to minimize the objective function (4.4), while keeping $v_j(k)$ invariant for $j \neq i$. Now the only terms in which v_i figures in (4.4) are

$$\left(|N_i| v_i - \sum_{j \in N_i} v_j - \sum_{j \in N_i} \hat{x}_{ij} \right)^2 + \sum_{j \in N_i} \left(|N_j| v_j - \sum_{\substack{k \in N_j \\ k \neq i}} v_k - \sum_{\substack{k \in N_j \\ k \neq i}} \hat{x}_{jk} - v_i - \hat{x}_{ji} \right)^2 .$$

When we perturb v_i to $v_i + \delta_i$, we get

$$\left(|N_i| (v_i + \delta_i) - \sum_{j \in N_i} (v_j + \hat{x}_{ij}) \right)^2 + \sum_{j \in N_i} \left(|N_j| v_j - \sum_{\substack{k \in N_j \\ k \neq i}} (v_k + \hat{x}_{jk}) - (v_i + \delta_i) - \hat{x}_{ji} \right)^2 .$$

Differentiating with respect to δ_i, and setting to 0 gives,

$$|N_i| \left(|N_i|(v_i + \delta_i) - \sum_{j \in N_i} (v_j + \hat{x}_{ij}) \right) - \sum_{j \in N_i} \left(|N_j| v_j - \sum_{k \in N_j} (v_k + \hat{x}_{jk}) - \delta_i \right) = 0.$$

Let

$$\begin{aligned} e_j &:= |N_j| v_j - \sum_{k \in N_j} (v_k + \hat{x}_{jk}) \\ &= \text{“Reported error of node j”}. \end{aligned}$$

Then

$$\left(|N_i|^2 + |N_i| \right) \delta_i + |N_i| e_i - \sum_{j \in N_i} e_j = 0.$$

So

$$\begin{aligned} \delta_i &= \frac{\sum_{j \in N_i} e_j - |N_i| e_i}{|N_i|^2 + |N_i|} \\ &= \frac{1}{|N_i| + 1} \left[\frac{1}{|N_i|} \sum_{j \in N_i} (e_j - e_i) \right]. \end{aligned}$$

So the distributed algorithm is very simple: At each iterate, some node, say node i, changes its v_i to $v_i + \delta_i$, where

$$\begin{aligned} \delta_i &:= \frac{1}{|N_i| + 1} \left[\frac{1}{|N_i|} \sum_{j \in N_i} (e_j - e_i) \right] \\ &= \frac{1}{|N_i| + 1} \left[\text{Average of } (e_j - e_i) \text{ reported by its neighbors } j \right]. \end{aligned}$$

Thus, sporadically, each node i broadcasts to its neighbors (i, v_i, e_i). From this each

node can calculate:

$$e_j = |N_j|v_j - \sum_{i \in N_j}(v_i + \hat{x}_{ji}).$$

It then adjusts its v_j to $v_j + \delta_j$. Then it rebroadcasts e_j, etc. ■

Now, if the clocks do not run at the same speed, that is, they have a drift or skew, a similar process would also have to be executed simultaneously to estimate the clocks' skews. If we substitute $log(\hat{\alpha}_{ji})$ for $\hat{x}_{ji}$ we are able to use the same solution just discussed to estimate skews while taking global constraints into account.

Also, due to the difference in skews between the nodes' clocks, every time the error is computed by a node i for the offset estimation, the v_j's and $\hat{x}_{jk}$'s of all its neighboring nodes k should be adjusted using the latest estimation of their skew in a way similar to the one shown in equation 3.13.

One can also obtain an alternative algorithm for the problem of minimizing (4.1). If we take another look at the original problem formulation in (4.1), we see that since we are trying to minimize v, we can simply use the following recursion:

$$v_i = \frac{\sum_{j \in N_i} v_j + \sum_{j \in N_i} \hat{x}_{ij}}{|N_i|} \text{ for all } i. \tag{4.5}$$

Therefore we have another distributed algorithm which is even simpler. Each node j simply broadcasts to its neighbors (j, v_j) sporadically, and from this, each node i can calculate its v_i following (4.5).

A further modification is to only take the average in (4.5) with respect to nodes that are at a fewer number of hops in distance from the reference node.

In [72], Giridhar and Kumar have analyzed the least-squares approach used in this protocol by relating the optimization problem given by (4.1), to the problem of determining the electrical resistance between two nodes in an electrical network

(in a similar way that Karp et al. did in [73] for RBS). They have shown that the performance of the least-squares algorithm, measured in terms of maximum error variance, is substantially better than that of a tree-based approach like TPSN, and also provide the lower and upper bounds on the settling time of the distributed algorithm we have just described.

4.2 Implementation on Berkeley Motes

The algorithm represented by (4.5) was implemented using MICA2 motes on a 40-node network where different topologies were enforced by software. That is, although all nodes operated on the same cell, packets were filtered out according to the multi-hop topology desired, i.e., as in MAC filtering.

4.2.1 Setup

As with the implementation of the pair-wise synchronization algorithm, we had a mote acting as a gateway to the PC that continuously collected the packets transmitted by the motes in the network. Each node sends a clock synchronization message to its neighbors, with a time interval randomly selected between 25 and 45 seconds so as to reduce the probability of collisions. The gateway mote requests the reference time each mote has estimated, every 15 seconds. To avoid collisions and errors in the measurements, instead of each mote responding to the queries made by the gateway mote immediately, they store the information regarding their estimates of the time at the reference, at the time each query is received on the external 512 KBytes flash EEPROM. The complete information was collected once the experiment was finished, so we could determine the accuracy of the algorithm by comparing the estimate the reference mote had made at each of the queries, which are identified using a sequence number, with the estimate that each of the other motes had at that same query.

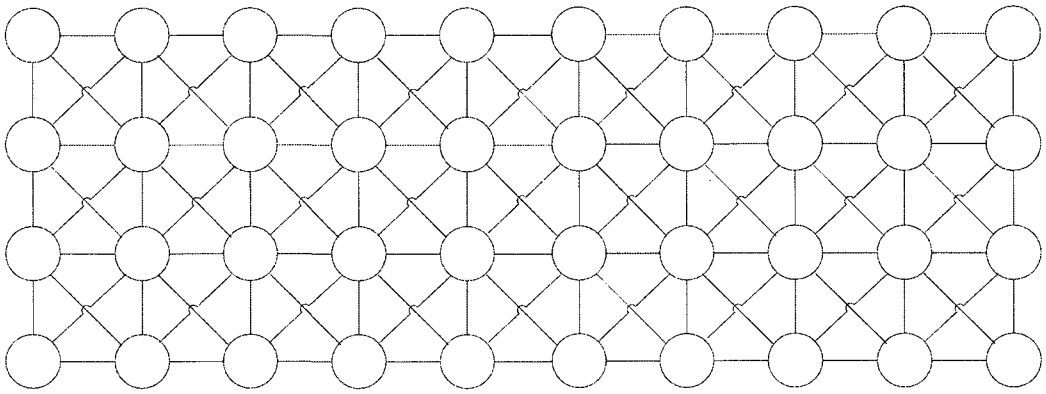


Figure 4.2: Link topology for the multihop implementation.

4.3 Evaluation

For the topology of 4x10 motes shown in Figure 4.2, which results in a maximum 9-hop network, we show the minimum, maximum and average accuracy obtained between the estimated time and the actual time at the reference across the whole network in Figure 4.3. The variant shown is that where the averaging is done over neighbors at a fewer number of hops from the reference node.

Such results are quite similar to the ones reported for FTSP in [17]. So taking into account that leading research groups on sensor networks, such as the ones in the University of Virginia and Ohio State, use FTSP, we set to comparing the behavior of this algorithm with FTSP, by running it on the exact same topology. FTSP nodes send clock synchronization messages every 30 seconds, and our gateway mote queries the global time that every mote has estimated every 15 seconds.

The minimum, maximum and average accuracy obtained between the estimated time and the actual time at the reference across the whole network through FTSP are shown in Figure 4.4.

Also plotted in Figure 4.5 is the behavior of the average accuracy as well as the standard deviation for each of the 9 hops in the network. We can observe that our algorithm provides better accuracy, and the estimates have smaller standard devia-

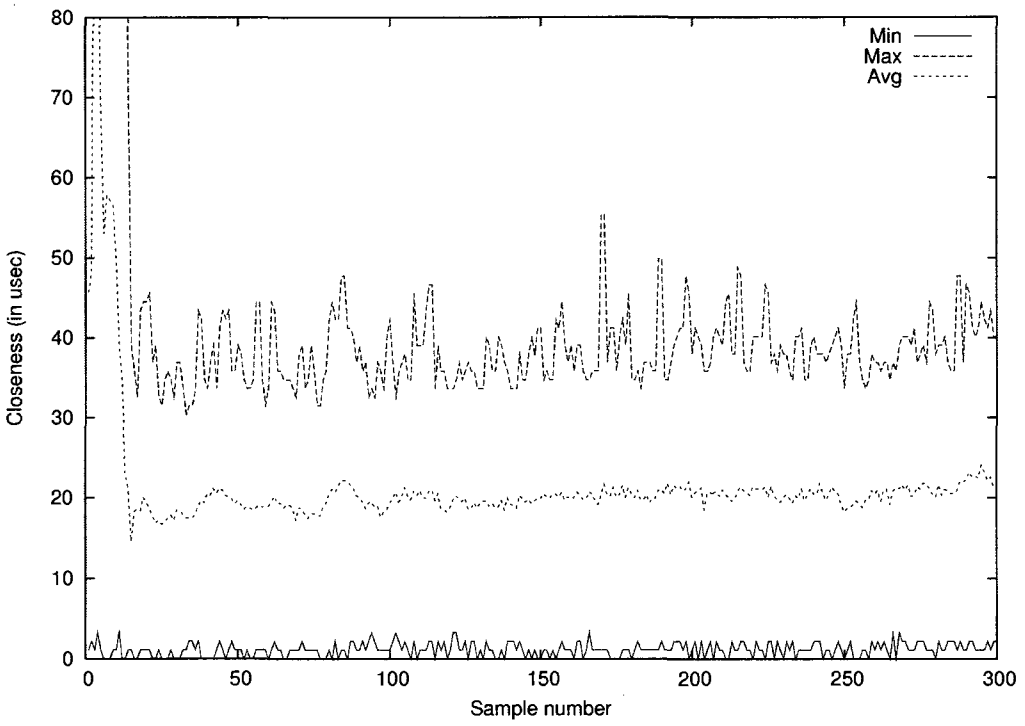


Figure 4.3: Average closeness of estimated to real time in a 40-node network.

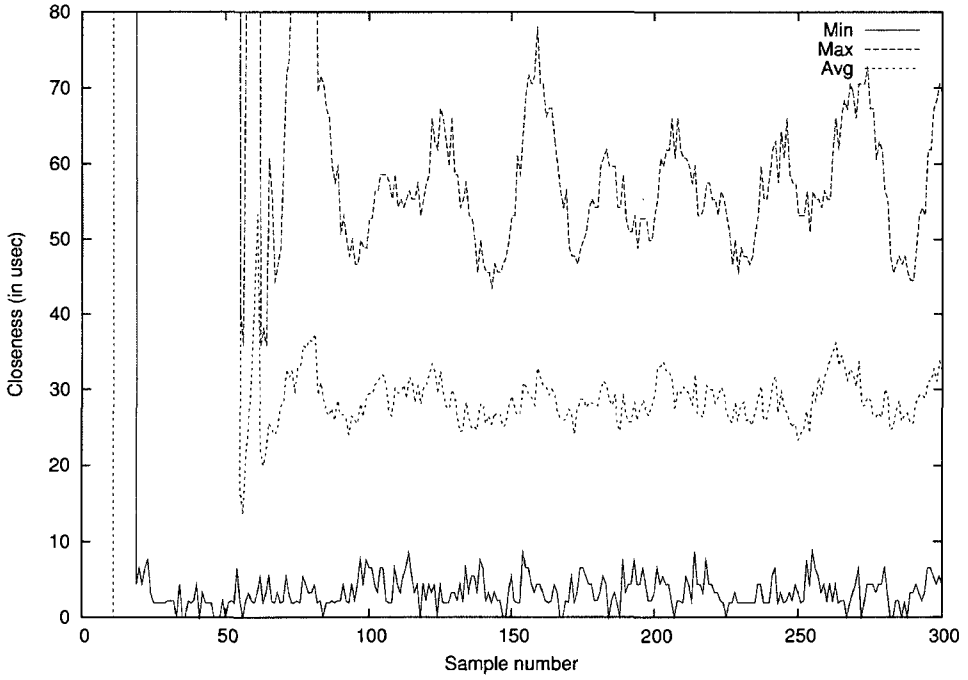


Figure 4.4: Experimental results of FTSP on the 40-node network.

tion, than the estimates obtained using FTSP in the same network. The statistics were computed starting at query 100 so they are not affected by the initial state,

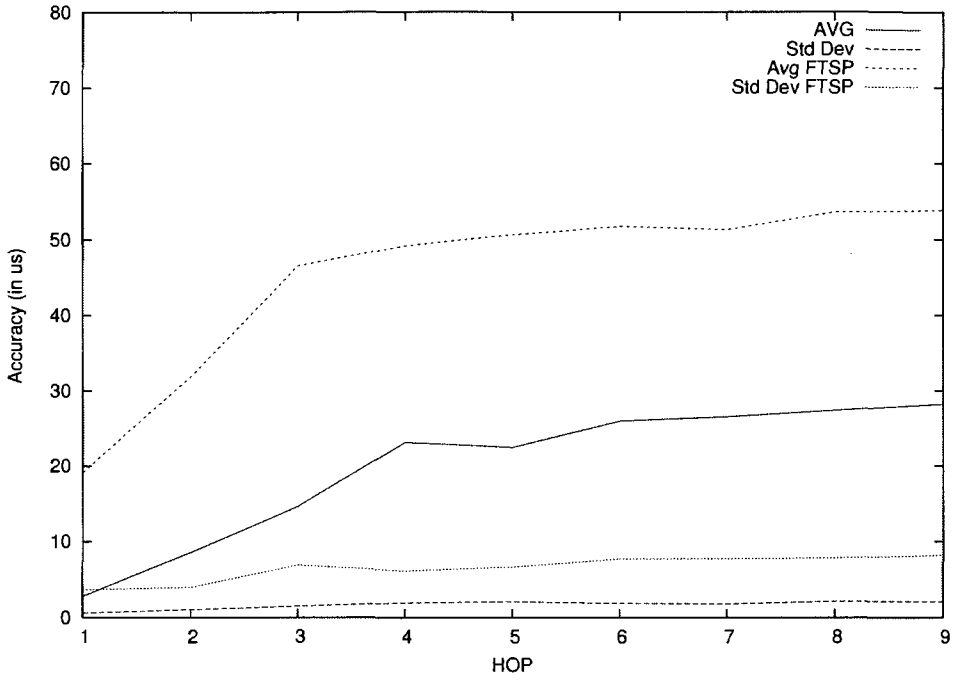


Figure 4.5: Comparison to FTSP.

which cannot be compared exactly since our algorithm starts with the reference node already determined, while FTSP goes through an election process to determine what node is the reference, i.e., the node with smaller ID.

CHAPTER 5

SECURITY AND CLOCK SYNCHRONIZATION

In the previous chapter we presented a completely distributed and asynchronous algorithm to achieve clock synchronization, where nodes communicate only with their neighboring nodes, and which gives a better performance in terms of accuracy and error variance when compared with a widely used protocol such as FTSP. However, like the other protocols mentioned in Section 2.5, it assumes a benign environment, that is, all nodes are cooperative and trustworthy.

It has been noted in Chapter 2 that clock synchronization is, in fact, a critical service in many distributed applications. In an adversarial environment, its functionality could be adversely affected by attackers, which would ultimately render the clock synchronization service useless. For instance, a malicious node could get control of a link between two nodes and change the characteristics of the link, causing the nodes to obtain faulty time estimates. This would then prevent the application using the clock synchronization from working as it should, e.g., objects would not be tracked or localized correctly, highway traffic would not be estimated adequately, etc. Therefore, there is a need to develop protocols to provide a *secure* clock synchronization capable of running in hostile environments and detecting the presence of attackers. In the next two chapters, we present our work on secure clock synchronization. We first present the results for a single link, and then on a network-wide scale. We begin by discussing in this chapter some required background to better understand the attacks that can be inflicted on a wireless sensor network, and how can we develop defenses against such attacks.

5.1 Requirements for Sensor Network Security

In order to prevent most of the attacks, a wireless sensor network should be designed to meet the following requirements [74]:

1. Data confidentiality. A node should keep sensitive data secret.

2. Data authentication. A node should be able to verify that a received message was actually sent by the claimed sender.

3. Data integrity. A node should be able to detect alterations in the received messages.

4. Data freshness. A node should be able to determine if a received message is recent, and not a replayed one.

To achieve data confidentiality, the standard approach is to use cryptographic techniques which could be symmetric (DES, RC5, or AES), or asymmetric (RSA, ECC) [75]. Data authentication is achieved by computing a message integrity code (MIC) using a private key (again, cryptography of some kind is involved) [76]. Data integrity can be achieved through data authentication. Data freshness is achieved by inserting a nonce (randomly generated value) or another time-related counter in the messages [77].

5.2 Attacks

There are several possible attacks on a wireless sensor network, which could be categorized as follows [78]:

1. Eavesdropping. The attacker seeks to determine what is the information being exchanged in the network. The attack could be passive, meaning that the

eavesdropper only listens to the messages, or active, meaning the eavesdropper sends queries in order to elicit further message exchanges.

2. Disruption. The attacker seeks to stop the application from working correctly. It does this by injecting messages, corrupting data, or by directly manipulating the environment.

Specific types of attacks include [79]:

- Denial of service attack. Its goal is to make a service unavailable to the clients. It could be performed by jamming, which is the transmission of a signal that interferes with the radio frequencies used by the radio [80], by violating the communication protocol at the link layer level, by refusing to route messages, or by flooding.

- Sybil attack. This is defined as a "malicious device illegitimately taking on multiple identities" [81]. This is an effective attack against routing algorithms, data aggregation, voting and fair resource allocation.

- Node replication attack. An attacker adds a node to an existing sensor network by replicating the node ID of an existing sensor node [82] leading to a disruption of the performance of the network.

- Attacks against privacy. Sensitive information could be discovered by the attacker by means of eavesdropping or traffic analysis [83].

- Wormhole attack. In this attack, the attacker receives a message at one location in the network, tunnels it to another location and replays it at the second location, introducing links that do not have the properties of the network (i.e., lower delay, higher bandwidth) [84].

- Man-in-the-middle attack. In this attack, the attacker intercepts messages exchanged between two nodes and relays such messages in a way that makes the two nodes believe the link they share is valid, while in reality the characteristics of the link have been modified [1].

- Physical attack. When deployed in hostile outdoor environments, given that the sensors are small, unattended and distributed, physical attacks resulting in physical damages to the node are highly likely [85, 86].

5.3 Defenses

Cryptography is the standard defense against eavesdropping, injection and corruption of messages. The different cryptographic methods require the establishment of a key (or keys) among the nodes in the network so they can perform the necessary operations to encrypt, decrypt or authenticate the messages exchanged. Symmetric key cryptography uses the same key to encrypt and decrypt a message. Asymmetric or public key cryptography uses different keys for encryption and decryption and provides for a simpler way to achieve message authentication. While symmetric key cryptography requires less computation resources, it involves a more complicated key distribution scheme.

In terms of cryptography and key distribution, some of the most important protocols developed are:

- TinySEC [87] is the first fully-implemented link layer security architecture for wireless sensor networks. Its goal is to guarantee data confidentiality, data authentication and data integrity. It was implemented under TinyOS. Its choice of block cipher is Skipjack, which the authors found to be the most suitable for the hardware available at the time (MICA2). TinySEC always authenticates

messages, and for this, it uses CBC-MAC to compute and verify the MICs. It relies on a single network-wide key, which is its main drawback.

- μTESLA [74] offers an asymmetric cryptographic type of broadcast data authentication using symmetric keys. The basic idea is to divide the time into intervals of equal duration and use a different key to compute the MICs in each interval. The disclosure of the key used in a given interval is delayed to a later interval. It requires the nodes to be loosely clock synchronized.

- LEAP [88] (Localized Encryption and Authentication Protocol) is a key management protocol for sensor networks. It supports four types of keys for each sensor, which the authors claim are adequate for all types of communications in sensor networks. Sensors are preloaded with an initial key from which further keys can be established.

- PIKE [89] is a protocol to establish a key between two sensors using as a trusted intermediary a third node located somewhere within the sensor network.

- Probabilistic key pre-deployment scheme [90] consists of three phases: key predistribution, shared-key discovery, and path-key establishment. In this work it is shown that when nodes randomly draw and store a small number of keys from a large pool, there is a considerably large probability that two neighboring nodes will have a shared key.

For attacks such as denial of service, node replication, Sybil, and wormhole attacks, several defenses have been proposed:

- For jamming at the physical layer, the standard defense involves various forms of spread-spectrum communication [77]. To handle jamming at the MAC layer, Wood and Stankovic [80] propose the utilization of a MAC admission control that is rate limiting.

- For node replication attacks and Sybil attacks, Newsome et al. [81] propose several defenses including wireless network testing, key space verification, and central node registration.

- For wormhole attacks, Hu et al. [84] propose two different types of defenses, geographic leashes, which use location information, and temporal leashes, which use timestamps, in order to constrain wormholes to a small region. In [91], Eriksson et al. propose a timing based defense called TrueLink. It does not rely on precise clock synchronization, and uses a modified medium access protocol. It is meant to be used together with a secure routing protocol.

5.4 Secure Clock Synchronization

Regarding clock synchronization, there are studies [92,93] describing possible attacks that could be performed on some of the clock synchronization protocols mentioned in Section 2.5. They can be summarized as follows:

- For RBS, a compromised node could send falsified synchronization information to its neighbor during the exchange period leading to an incorrect estimation of the phase and skew by the honest node.

- For TPSN, a compromised node can affect its children by sending them incorrect timestamps, or it could also lie about its level in the tree.

- For FTSP, a compromised node could claim to be the root node with ID 0 and begin with a higher sequence number than the actual root node making all nodes disregard the updates sent by the actual root node.

Very few secure clock synchronization algorithms have been proposed as defense against these attacks.

5.4.1 SPS and SGS

In [94], using TPSN as a base, Ganeriwal et al. propose the Secure Pairwise Synchronization (SPS) protocol to handle a *pulse-delay attack*, where an attacker delays the time at which a synchronization packet is received, modifying the offset and delay calculation performed at the receiver (Section 2.5.5 shows how these calculations are made). SPS provides authentication to MAC layer timestamping by adding a timestamp and message integrity code (MIC) as the message is being transmitted, which works only for sensors with low data rates such as MICA2, but not with higher data rates as the ones present in more recent sensors such as the IMote2. SGS (Secure Group Synchronization) uses SPS to synchronize a group of nodes, and assumes all nodes are within range.

5.4.2 Secure and resilient clock synchronization

In [95], Sun et al. propose two statistical techniques for secure and resilient clock synchronization. The level-based clock synchronization technique is targeted at static sensors, and constructs a tree hierarchy where clock synchronization messages flow from the root to the leaves. The diffusion-based clock synchronization allows sensor nodes to synchronize to the common source through any neighbor nodes without requiring any hierarchal structure. The goal of both techniques is to provide fault-tolerance up to a specific number of malicious nodes, but they are not able to handle pulse-delay and wormhole attacks since they have no way to authenticate the timeliness of the synchronization messages. The techniques were improved in [96], and a new approach called TinySeRSync emerged, using authenticated MAC layer timestamping for a single hop synchronization, and μTESLA local broadcast authentication protocol for global synchronization.

CHAPTER 6

SECURE CLOCK SYNCHRONIZATION OVER A SINGLE LINK

We will now present a secure clock synchronization protocol that is able to detect man-in-the-middle attacks using only timing information under appropriate conditions. We begin with a thorough theoretical description of the protocol [1], followed by a description of the implementation that has been performed using IMote2 motes and TinyOS 2.1, as well as the results of our experimental evaluation.

6.1 Protocol Description

6.1.1 Notation

Throughout this chapter we will be using the following notation

- τ_A denotes the turn-around time of node **A**, which is the context switch time taken by a radio transceiver to switch from transmitting to receiving, or vice-versa
- m_{AB_n} denotes the delay induced by attacker on the n-th packet from **A** to **B**
- o_A denotes the clock offset of node **A**, relative to a reference or master clock t
- S_A denotes the clock skew of node **A**, relative to a reference or master clock t
- S_{AB} denotes the relative clock skew of node **B** with respect to node **A**, $S_{AB} = S_B/S_A$

- δ_{AB} denotes the one-way delay from **A** to **B**, which includes only the propagation delay between nodes **A** and **B**
- r denotes the round-trip time, which is the sum of δ_{AB} and δ_{BA}
- r' denotes the sum of the round-trip time r and the delay induced by the attacker in both directions
- $\rho_{AB}(n)$ denotes the local clock time at node **B** when the n-th packet from **A** is received
- $\sigma_{AB}(n)$ denotes the local clock time at node **A**'s when it sends its n-th packet to **B**
- $P_{AB}(n)$ denotes the n-th packet sent from **A** to **B**

6.1.2 Assumptions

- Clocks are affine. An affine clock is one where the local time at node **A** has the value

$$A(t) = S_A t + o_A\,,$$

where

t is the time at some master clock,

$S_A > 0$, denotes the *clock skew*, which is the relative speed of the clock at node **A** with respect to the master clock, and

o_A denotes the *offset* of node **A** with respect to the master clock.

- Nodes can accurately timestamp packets, which, as we saw in Section 3.4, is possible for MICA2 motes. In Section 6.2.2, we will see how this is possible in IMote2 motes.
- If two nodes can communicate directly we say they share a link.

- Both endpoints of a link are half-duplex nodes, meaning they can only receive or transmit, but not both concurrently. In this case τ_A is the time to switch from transmit to receive, or vice-versa.

- Nodes sharing a link share a private key, which is used to provide authentication and confidentiality. Using the private keys in this way eliminates traditional, cryptographic man-in-the-middle attacks. Thus, nodes exchange unpredictable and verifiable messages. A number of techniques exist for deriving the private key [97].

- The attacker can be either half-duplex, as the endpoints are, or full duplex, meaning it has the ability to transmit and receive concurrently.

- If the attacker is half-duplex, then the attacker's radio has a negligible turn-around time, meaning it can switch from receiving to transmitting or vice-versa practically immediately, with zero context switch time.

- The attacker is unable to break the cryptography used in the protocol.

- The attacker is free to do as it pleases, but desires to remain undetected.

- The attacker is free to use additional hardware such as directional antennas, emissions detection hardware, etc.

6.1.3 Basic clock synchronization protocol

The basic clock synchronization for two nodes **A** and **B**, is achieved by exchanging 4 packets as shown in Figure 6.1. $P_{AB}(n)$ denotes the n-th packet sent from node **A** to **B** and includes the time at which it is sent, $\sigma_{AB}(n)$, as well as the last time at which a packet was received in the reverse path, $\rho_{AB}(n-1)$. Once these 4 packets have been exchanged, node **A** has enough information to estimate the relative skew

of **B** with respect to **A**, S_{AB}, as well as the one-way delay from **A** to **B**, δ_{AB}, using the following equations:

$$S_{AB} = \frac{\rho_{AB}(n) - \rho_{AB}(n-1)}{\sigma_{AB}(n) - \sigma_{AB}(n-1)}, \tag{6.1}$$

$$\delta_{AB} = \rho_{AB}(n) - S_{AB}\sigma_{AB}(n). \tag{6.2}$$

Once another packet is sent from **A** to **B**, $P_{AB}(3)$, **B** is able to make similar measurements in the reverse path.

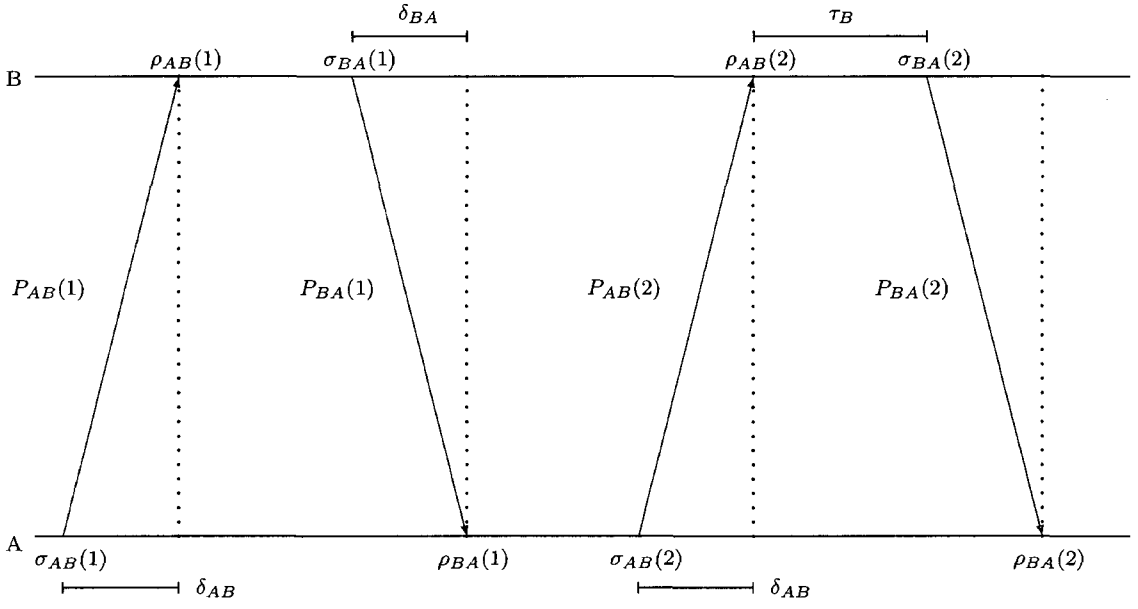


Figure 6.1: Message exchanges to achieve basic clock synchronization

Freris and Kumar [50] show that after the completion of this basic clock synchronization, legitimate nodes can exactly predict the time at which the packet $P(n+1)$ will be received according to the receiving node's clock by using the following prediction equation:

$$\hat{\rho}_{AB}(n+1) = S_{AB}\sigma_{AB}(n+1) + \delta_{AB}. \tag{6.3}$$

6.1.4 Adding security to the protocol

By using the basic clock synchronization protocol in the previous section, legitimate nodes are able to impose restrictions on the kind of additional relaying delays a man-in-the-middle attacker can impose on the packets exchanged between them. These restrictions are described in the following theorem.

Theorem 6.1.1. *If two nodes having affine clocks use a fixed network where the delay between the nodes is constant, and are able to authenticate packets sent by each other, then any attacker that delays packets by an amount of time that is not constant, can be eventually detected using timing information alone.*

Intuition: We will refer to the time given by the sender's clock at the time at which it begins transmitting a packet (refer to Figure 2.3 on page 13) as the sender timestamp, and the time according to the receiver's clock at which it begins receiving the packet as the receiver timestamp. In the absence of an attacker, the receiver timestamps for messages in a node **B** are affine in the sender timestamps by node **A** at the transmission of such messages. Thus, if an attacker **M** wants to remain undetected, the delay m_{AB_i} it induces in message i must still result in a reception timestamp affine in the sending timestamp. Node **B** can calculate its relative skew with respect to **A**, S_{BA}, using these timestamps. If the calculated skew is lower than the actual skew then the delay the attacker **M** induces is decreasing, and once this induced delay reaches zero, the attacker **M** will be detected since it cannot relay a message before receiving it, i.e., it cannot induce negative delays. If we also use the fact that the product of skews in both directions must be 1 ($S_{AB}S_{BA}$=1), then **M** is not able to increase the induced delay in either direction without being detected. Therefore, **M** can impose only a constant delay that is the same for all packets if it wants to remain undetected.

Proof. Let $a_1, a_2, \ldots, a_k$ be the transmission times of packets sent by **A** to **B**, and

$b_1, b_2, \ldots, b_k$ be the transmission times of packets sent from **B** to **A**. All transmission times are according to the master clock t. Let us define δ_{AB} to be the normal (constant) delay from **A** to **B**, and m_{AB_i} to be the delay the attacker additionally imposes on the ith packet from **A** to **B**. Let us also define the affine transformation functions $A(t) = S_A t + o_A$ and $B(t) = S_B t + o_B$ that receive the time from the master clock t as input, and return the time as measured by nodes **A** and **B**, respectively.

Node **B** calculates the difference between the send and receive timestamps of packet i as

$$d_i = B(a_i + \delta_{AB} + m_{AB_i}) - A(a_i).$$

If we use the expansion $B(a_i + \delta_{AB} + m_{AB_i}) = B(a_i) + S_B(\delta_{AB} + m_{AB_i})$, and then we add and subtract $S_{AB}A(a_i)$, we get

$$d_i = S_{AB}A(a_i) - A(a_i) + B(a_i) - S_{AB}A(a_i) + S_B(\delta_{AB} + m_{AB_i}).$$

This can be simplified using

$$\begin{aligned} B(a_i) - S_{AB}A(a_i) &= (S_B a_i + o_B) - (S_B a_i + S_{AB} o_A) \\ &= o_B - S_{AB} o_A \,, \end{aligned}$$

to get

$$d_i = (S_{AB} - 1)A(a_i) + S_B m_{AB_i} + o_B - S_{AB} o_A + S_B \delta_{AB} \,.$$

Note that in the absence of the attacker, i.e., when $m_{AB_i} = 0$, then d_i is affine in a_i. Hence, for the attacker **M** to remain undetected, it must choose a m_{AB_i} such that d_i continues to remain affine in a_i.

From any two $d_i, d_j\,(i \neq j)$ and the corresponding send timestamps, $A(a_i), A(a_j)$, **B** calculates its *perceived skew* relative to **A** using

$$S'_{AB} = \frac{d_j - d_i}{A(a_j) - A(a_i)}.$$

Given that m_{AB_i} is affine, it must be either a constant, increasing, or decreasing function of $A(a_i)$. When m_{AB_i} is increasing, $S'_{AB} > S_{AB}$, and when m_{AB_i} is decreasing, $S'_{AB} < S_{AB}$.

In case m_{AB_i} is decreasing, after a large enough finite time, the affine delay m_{AB_i} will become negative, $m_{AB_i} < 0$, at which point the attacker would be caught due to violating causality. This forces the attacker to choose $S'_{AB} \geq S_{AB}$ to remain undetected. A similar argument for packets sent from **B** to **A** shows that for the attacker to remain undetected, it must also choose $S'_{BA} \geq S_{BA}$. Then, for the attacker to remain undetected it must choose $S'_{AB}S'_{BA} \geq S_{AB}S_{BA} = 1$, but when $S'_{AB}S'_{BA} \neq 1$, nodes can detect an attack since they can exchange their skew estimates and check to see if their product deviates from 1. Therefore, the only option left for the attacker, so it can remain undetected, is to choose $S'_{AB} = S_{AB}$, which implies that m_{AB_i} is constant. □

Therefore, to secure the basic clock synchronization protocol of Section 6.1.3, once nodes **A** and **B** are synchronized:

1. the sender **A** includes the estimation of the time at which the packet $n+1$ will be received, $\hat{\rho}_{AB}(n+1)$, calculated using (6.3),

2. the sender **A** appends a Message Integrity Code (MIC) to the packet contents, which is calculated using a private key shared with the receiver,

3. the receiver **B** authenticates the packet contents,

4. the receiver **B** calculates the skew and delay using (6.1) and (6.2), and determines if the delay is consistent with prior delays. If not, the attacker is instantly detected.

If the man-in-the-middle attacker is able to induce only a constant delay, we can still detect it using only timing information under certain conditions which we now describe in the sequel.

To reduce the possibility of cryptographic attacks, a neighbor discovery subprotocol is used.

Detection of a half-duplex attacker

As previously mentioned, a half-duplex attacker is able to transmit or receive messages but not perform both concurrently. To detect such an attacker, we can take advantage of this limitation on concurrency. The protocol could send messages long enough for the attacker to not be able to delay all messages by a constant time, violating the synchronization requirements described above, and allowing its detection.

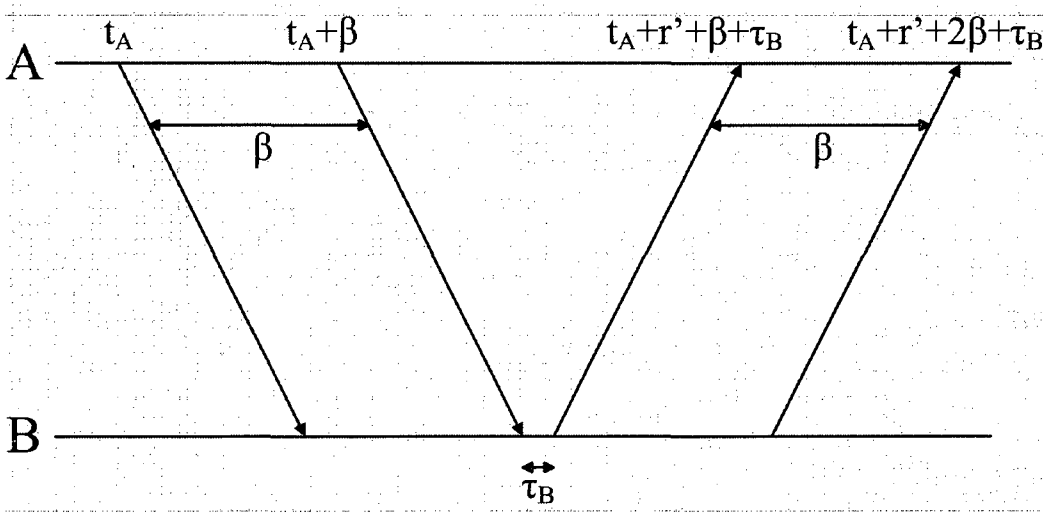


Figure 6.2: Detecting a half-duplex attacker. Source: [1]

To illustrate this protocol, we use Figure 6.2. We assume that node **A** synchronizes with node **B**, that the round-trip time is measured as r when there is no man-in-

the-middle attacker present, and that when a man-in-the-middle attacker becomes present between **A** and **B**, the round-trip time is measured as r'. As long as the delay induced by the attacker, $r' - r$ is a non-negative constant delay, the attacker remains undetected, by Theorem 6.1.1.

Once nodes **A** and **B** are synchronized, node **A** registers the current time in its clock, t_A, and sends a timestamped message $P_{AB}(n)$ of duration

$$\beta > \frac{1}{2}(r' + \tau_B)\,, \tag{6.4}$$

where τ_B is the turnaround time of node **B**. On reception of this message, node **B** switches from receiving to transmitting mode, and begins sending a timestamped message $P_{BA}(n)$ of the same duration β to node **A**.

Once node **A** receives the completes message $P_{BA}(n)$, it calculates the elapsed time, $T_{elapsed}$, from when it began the transmission of message $P_{AB}(n)$ to the completion of reception of message $P_{BA}(n)$, which is expected to be

$$T_{elapsed} = r' + \tau_B + 2\beta\,.$$

However, from (6.4), $r' + \tau_B < 2\beta$, and therefore,

$$T_{elapsed} = r' + \tau_B + 2\beta < 4\beta\,.$$

This leads to the following lemma.

Lemma 6.1.2. *The above detection protocol is able to detect a half-duplex man-in-the-middle attacker.*

Proof. Given that the attacker is half-duplex, it cannot transmit and receive concurrently. Therefore, the attacker requires at least 4 times the message transmission

duration, β, so that it will be able to buffer, delay, and forward both messages $P_{AB}(n)$ and $P_{BA}(n)$. This amount of time is required by the the attacker since it cannot predict when the messages $P_{AB}(n)$ and $P_{BA}(n)$ will be sent. This will result in an elapsed time $T'_{elapsed} \geq 4\beta$, which exceeds the time expected by node **A**, $T_{elapsed}$. Therefore the attacker is detected. □

6.2 Implementation

The protocols above described have been implemented on a testbed comprised of Crossbow's IMote2 sensor nodes [65], which include Zigbee compliant CC2420 radios capable of transmitting at a rate of 250 kbps [98]. The implementation is built on top of TinyOS 2.1 [60]. The first goal of the implementation is to determine whether the assumptions and properties of the theoretical results found in [1] do in fact hold in practice. This includes several verifications:

i) That practical clock drift will in fact show enough stability that delays can be estimated with reasonable accuracy.

ii) The accuracy in estimating delays in the absence of an attacker is sufficient that the product of the estimates of S_{AB} and S_{BA} is close to 1 to within a desirable accuracy.

iii) The scheme for catching a half-duplex attacker works for practical turnaround times.

The implementation consists of three major parts:

1. Exponential smoothing. It allows us to handle variability in the timestamps.

2. MAC layer timestamping. It allows to timestamp a packet at the actual time the radio begins its transmission or reception. See Figure 2.3.

3. Neighbor discovery subprotocol. It allows the nodes to detect replayed packets.

6.2.1 Exponential smoothing

Exponential smoothing is a linear regression procedure which assigns exponentially decreasing weights to the data being collected, as it becomes older. It helps us to handle the variability in the timestamps, which is the data we are collecting. We use exponential smoothing to estimate the clock skew, the one-way delay, and the predicted time at which a receiving node will receive a packet, as follows.

The estimate of the skew of **B** with respect to **A**, which is the ratio of the speed of the clock at **B** to the speed of the clock at **A**, made by **A** after the N-th packet from **B** to **A** is received, is estimated as

$$\hat{S}_{AB}(N) = \frac{\sum_{n=2}^{N} \gamma^{N-n} \frac{\rho_{AB}(n)-\rho_{AB}(n-1)}{\sigma_{AB}(n)-\sigma_{AB}(n-1)}}{\sum_{n=2}^{N} \gamma^{N-n}}, \tag{6.5}$$

where $0 < \gamma < 1$ is the exponential forgetting or smoothing factor, which in our implementation is assigned a value of 0.99, after the simulation results discussed in Section 3.3. The hat-notation indicates that the quantity under the hat is an estimate. As mentioned in Section 6.1.1, $\rho_{AB}(n)$ denotes the time on **B**'s clock at the time at which the n-th packet from **A** arrived, and $\sigma_{AB}(n)$ denotes the time on **A**'s clock when **A** sends its n-th packet to **B**.

The one-way delay from node **A** to node **B** is estimated as

$$\hat{\delta}_{AB}(N) = \frac{\sum_{n=3}^{N} \frac{\rho_{AB}(n)-\hat{S}_{AB}(n-1)\sigma_{AB}(n)}{\hat{S}_{AB}(n-1)} \gamma^{N-n}}{\sum_{n=3}^{N} \gamma^{N-n}}.$$

The time at which the n-th packet sent by node **A** will be received by node **B**,

according to the clock at node **B**, is estimated as (see (6.3))

$$\hat{\rho}_{AB}(n) = \hat{S}_{AB}(n-1)[\sigma_{AB}(n) + \hat{\delta}_{AB}(n-1)]\,. \tag{6.6}$$

Node **B** will similarly use exponential smoothing to arrive at the estimate of the skew in the other direction $\hat{S}_{BA}(N)$, the estimate of the one-way delay $\hat{\delta}_{BA}$, and the prediction of the time at which node **A** will receive the n-th packet sent by node **B**, according to node **A**'s clock, $\hat{\rho}_{BA}(n)$.

Now, if we were to use these equations as shown, the implementation would have to store all the collected timestamps, which is not suitable for a sensor with limited memory. However, we can apply the procedure described in Section 3.2 to derive a recursive method to calculate the smoothed estimates. By doing this, we only need to store the last timestamp obtained and the previous estimate value in our implementation.

6.2.2 MAC layer timestamping

In order to eliminate the variable processing delay in the network stack in our calculations, we have modified the MAC layer of TinyOS 2.1 to allow the timestamping of a packet at the time the first byte is sent, instead of doing it at the application layer; see Figure 2.3. We also record the time at which the first byte is received by the MAC layer at the receiver side for later use by the application layer, see Figure 2.3 again, in a way similar to what is described in Section 3.4.

To implement this in the new ChipCon CC2420 radio module found in the IMote2 sensor node, we use the ideas provided at the TEP 132 [99] and its reference implementation. The reference implementation timestamps packet transmission and reception in an Atmel RF230 transceiver using a 32KHz timer, but since we need microsecond resolution, we had to make the following changes in the TinyOS structure:

- a new module to provide a microsecond counter was built,
- the General Purpose I/O handler in charge of signaling the SFD interrupt was modified to use our microsecond counter instead of the 32KHz counter, and
- the CC2420 modules in charge of packet transmission and reception were modified to use our microsecond counter instead of the 32KHz counter.

The changes were carefully made so as to avoid disrupting other modules or applications in TinyOS.

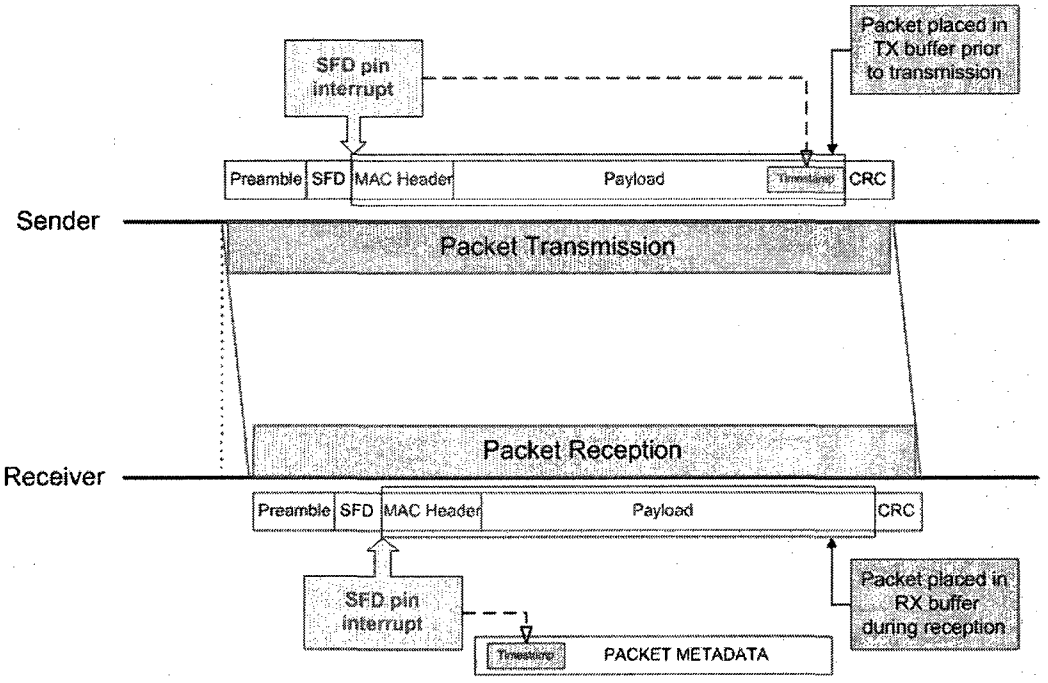


Figure 6.3: Illustration of MAC layer timestamping in the CC2420 radio.

The following process is illustrated in Figure 6.3. In the CC2420 [98], which is a packet radio (unlike the CC1000 byte radio found in MICA2), we need to place the message to transmit in the TX buffer prior to transmission; and once a packet is received, we read the contents from the RX buffer. Transmission begins with a preamble, which is used for radio calibration, followed by a Start Frame Delimiter (SFD). On the sending side, once this SFD is sent, and on the receiving side, once the

SFD is received, the SFD pin in the CC2420 radio goes active, which in turn triggers an interrupt in the IMote2. By modifying the corresponding interrupt service handler, we can either

- record the current time with the metadata associated with the packet on the receiving side, so that the upper layers are aware of the time at which the packet was received, or
- directly modify the TX buffer to insert the current timestamp in the packet that is currently being transmitted.

The idealized protocol described in Section 6.1.4, which inserts an estimated time of arrival (ETA) in each packet as calculated using (6.3), as well as an authenticator for the packet contents using a cryptographic scheme such as AES [100], cannot be achieved in our implementation. The reason is that we are timestamping the packets being sent at real time during transmission, and it takes 830 μs for the IMote2 to compute a floating-point multiplication and 4650 μs to compute an AES encryption, for instance. We therefore modified the way the ETA is calculated and the way the authentication is performed. In the case of the ETA, it is now performed by the *destination*, taking advantage of the fact that the information required for such task, i.e., skew and one-way delay estimates, are already included in the packet since they are necessary for the verification of skew consistency. As for the authentication, we decided on including the authenticator for the current packet being sent in the *next* packet. This approach has the drawbacks of increasing packet loss probability and latency, but we are now able to timestamp the packets in the way we need it.

6.2.3 Neighbor discovery subprotocol

We include a neighbor discovery subprotocol to make sure that an attacker cannot replay packets between sessions by ensuring the freshness of each clock synchronization

packet. Otherwise an attacker could replay old synchronization packets and make a node believe that a good link is not behaving correctly.

We assume that when two nodes **A** and **B** share a link, they also share a secret key, referred to as the *master key.* There are several schemes to obtain this master key: key pre-distribution schemes, pairwise key establishment, public key schemes, among others [97]. Instead of using the master key to perform the cryptographic operations, our protocols use a *session key* throughout a single session. A session starts when the neighbor discovery subprotocol is executed, and it ends when the two nodes stop being in each other's range or when the packet number space is exhausted in either direction. In the latter case, the neighbor discovery subprotocol is run again to establish a new session key.

To create a session, when node **A** believes that node **B** is its neighbor, the following exchange of packets takes place:

A $\rightarrow$ **B**: INITIATE, **A**, **B**, η_A, MIC(packet)$_{K_{AB}}$

A sends an INITIATE packet to **B** which contains the addresses of **A**, **B**, a nonce η_A, and an authenticator calculated using the master key K_{AB}.

B $\rightarrow$ **A**: RESPOND, **B**, **A**, η_A, η_B, MIC(packet)$_{K_{AB}}$

B generates its own nonce η_B to ensure that η_A corresponds to a fresh request from **A**, and sends **A** a RESPOND packet which contains the addresses of **B**, **A**, the nonce η_A, the nonce η_B, and an authenticator calculated using the master key K_{AB}.

A $\rightarrow$ **B**: ACCEPT, **A**, **B**, η_A, η_B, MIC(packet)$_{K_{session}}$,

If the RESPOND packet is received as a timely response to its INITIATE packet, it knows that η_B is fresh (because it is tied to η_A, so the pair (η_A,η_B) corresponds to a fresh session). **A** then generates the session key $K_{session}$=H($\eta_A||\eta_B||K_{AB}$) and returns an ACCEPT packet containing **A** and **B**'s addresses, both nonces η_A and η_B, and an authenticator calculated using the new session key $K_{session}$.

If **B** receives the ACCEPT packet as a timely response to its RESPOND packet,

it knows that η_A is fresh (because it is tied to η_B, so the pair (η_A,η_B) corresponds to a fresh session). **B** can then compute the session key $K_{session}$=H($\eta_A||\eta_B||K_{AB}$) and can proceed to send clock synchronization packets to **A** using the new session key $K_{session}$.

6.3 Evaluation

We now present the results from our implementation, as described in the previous section. First we show that the clock synchronization assumptions hold and that we can predict packet arrival times at remote nodes with very good accuracy.

6.3.1 Clock synchronization

The full clock synchronization algorithm was implemented using the MAC layer timestamping described in Section 6.2.2. We run several experiment involving two nodes **A** and **B**, where nodes exchange messages in the way depicted in Figure 6.1, every 3 seconds. The packets sent from node **A** to node **B** contain the following information:

- Packet number. Used to identify the clock synchronization packets and for the neighbor discovery subprotocol, which is run once the packet number wraps around, to ensure the freshness of every packet.
- Acknowledgment number. Used to refer to the most recent packet received in the reverse direction.
- $\rho_{BA}(n-1)$. Time at which the last packet (n-th packet) sent by **B** was received by **A**.
- $\hat{\delta}_{AB}$. The current estimate of the one way delay from **A** to **B**.

- $\hat{S}_{AB}$. The current estimate of the skew of **B** with respect to **A**, as an IEEE 754 double-precision floating point value.

- MIC($P_{AB}(n-1)$). Authenticator of the previous packet sent. It is calculated using HMAC [76], with SHA1 [101] as the hash function.

- $\sigma_{AB}(n)$. Time at which the current packet was sent. This is actually the last field in the packet in order for the interrupt service handler for the SFD pin to have enough time to place the timestamp once the SFD interrupt is triggered.

6.3.2 Validation of clock synchronization

The clock synchronization behavior depends on two assumptions:

1. The product of skews $S_{AB}S_{BA} \approx 1$, even when the two are independently measured.

2. Our timestamps are sufficiently accurate to detect small perturbations in the RTT.

To validate these assumptions, we ran the protocol using two motes with no attackers present. Samples were collected from over 1600 runs of the synchronization protocol. Each sample corresponds to one exchange of packets: one packet from node **A** to **B** and one packet from **B** to **A**. We examined the behavior of the skews, the product of the skews, and the accuracy of our calculations of the ETA.

Figure 6.4 shows the results obtained from calculating skew. We subtracted 1 from each of the skew values in order to allow the y-axis labels to be meaningful. We can observe that even though each skew is measured independently by each of the nodes, and even though the skews drift over time, the product of the measured skews stays very close to 1, confirming the first assumption mentioned above.

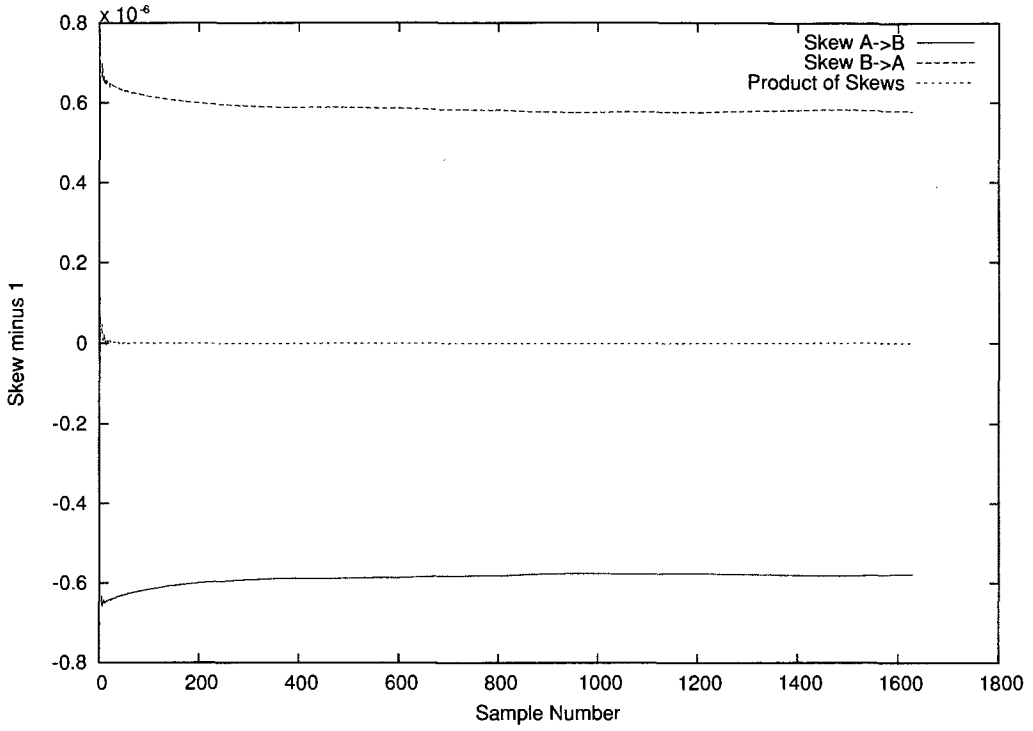


Figure 6.4: Linear skew

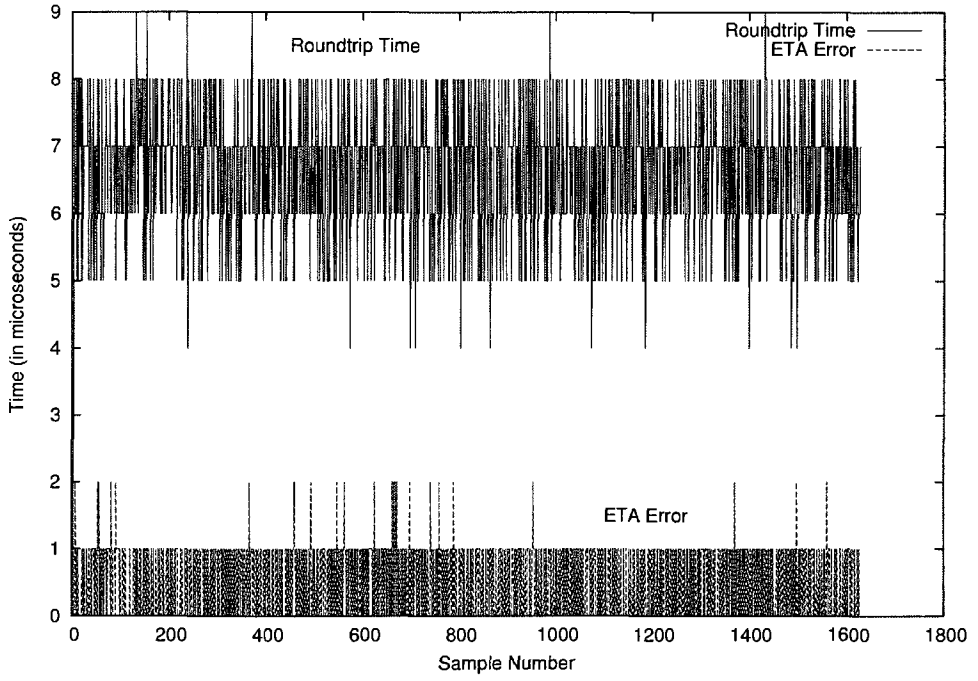


Figure 6.5: Accuracy of packet arrival time prediction and RTT behavior

Figure 6.5 shows the accuracy of our ETA calculations and the RTT behavior over time. The median error for the data shown is 0 μs and the average error is 0.5 μs,

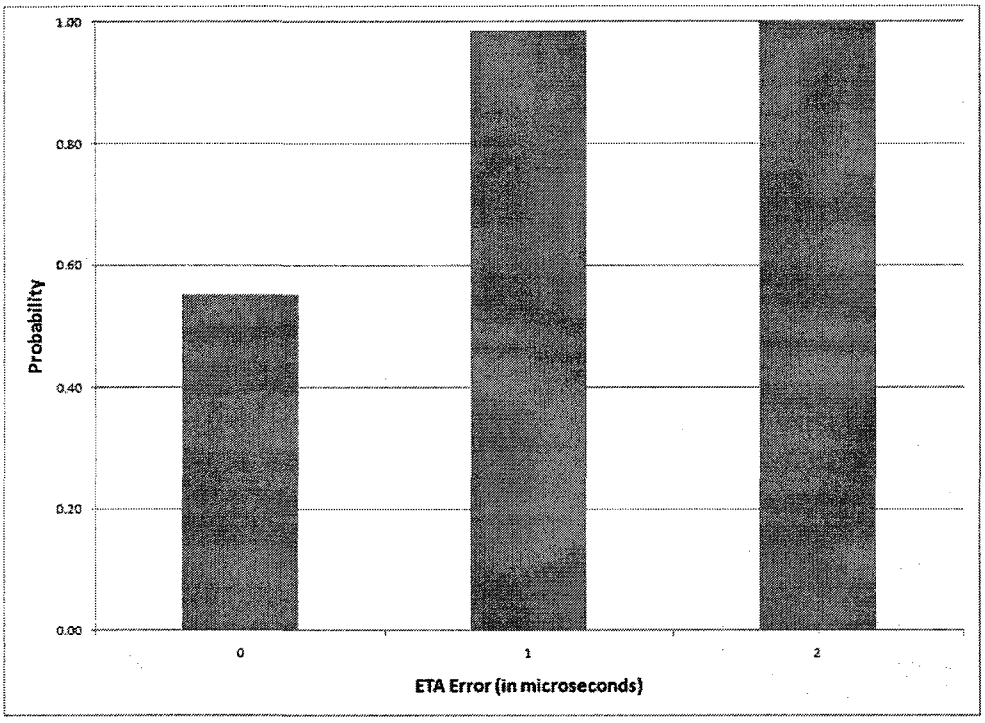


Figure 6.6: CDF of the errors in the arrival time prediction

indicating a high level of accuracy given that the clocks we used had 1 μs resolution. For a better perception of the accuracy obtained, Figure 6.6 shows the cumulative distribution function of the errors observed. This result shows that nodes are able to accurately predict the time at which a receiver will receive a packet according to the receiver's clock, and validates the second assumption.

6.3.3 Man-in-the-middle attacker

To test our clock synchronization protocol under the influence of man-in-the-middle attackers, we implemented the man-in-the-middle functionality in a mote.

Setup

The setup, which is shown in Figure 6.7, consists of nodes **A** and **B**, a man-in-the-middle attacker (MITM) (which simply relays all packets heard with a higher

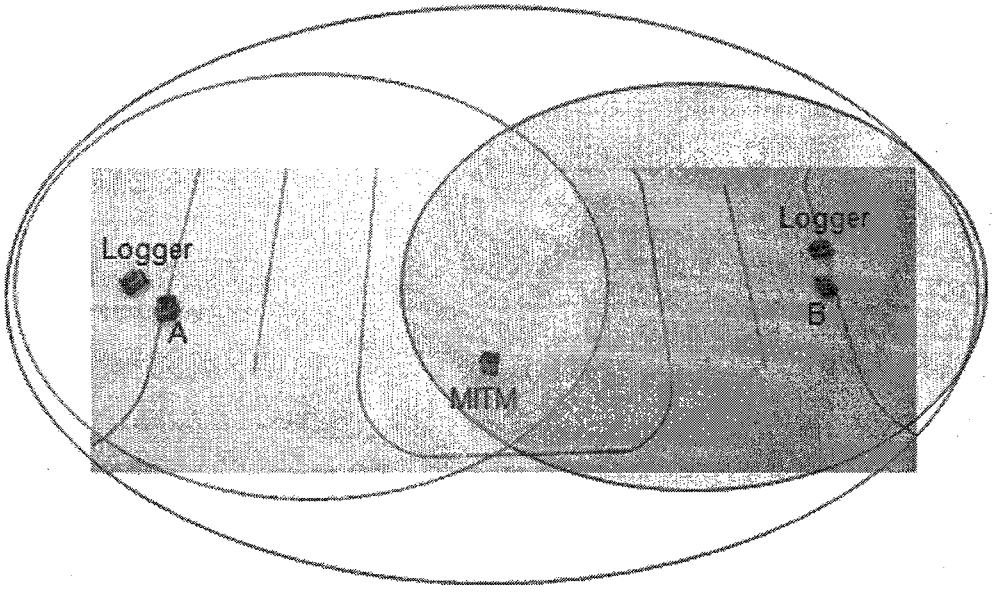


Figure 6.7: Experimental setup with an MITM attacker

transmission power, this relaying is done at the lowest possible level in the network stack of TinyOS in order to reduce the variability in forwarding latency), and two observer nodes that log all traffic they hear for later analysis.

The observer nodes are used in order to avoid impacting the calculations performed by nodes **A** and **B**, one is placed near node **A** and the other one near **B**. Nodes **A** and **B** are configured to transmit at a very low transmission power so that it is the MITM who connects **A** and **B** even though they are incapable of direct communication with each other.

Validation of skews with a man-in-the-middle attacker

In the idealized protocol, we argued that for an attacker to remain undetected, it needs to delay packets only by a constant amount of time. We now proceed to validate this by measuring the skews when a man-in-the-middle attacker is present. If the attacker introduces a non-constant delay with sufficient variation, the skews would be affected and the attacker would be detected. For this validation, we ran our

clock synchronization protocol on the link created by a man-in-the-middle attacker, which as we have just described, simply relays all packets heard.

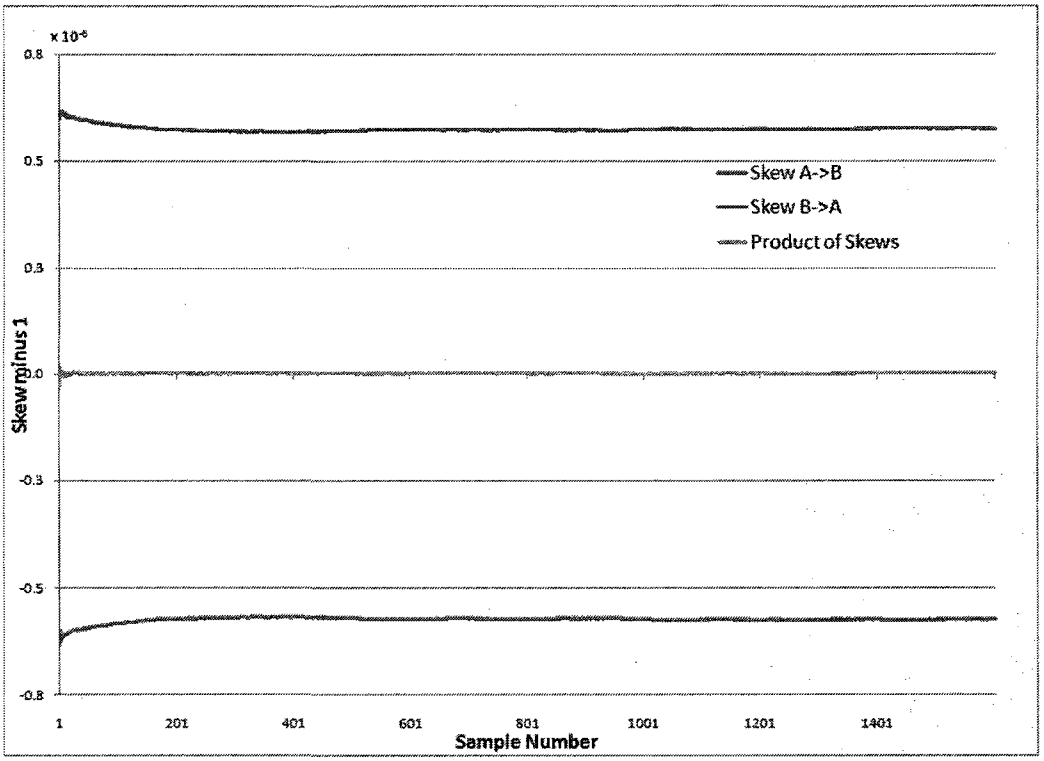


Figure 6.8: Skews observed with an MITM attacker present

The resulting skews are plotted in Figure 6.8. Once again, we subtracted 1 from each of the skew values to allow the y-axis labels to be meaningful. We can observe that the product of skews still stays very close to 1, although not as close as the ones observed when there is no man-in-the-middle attacker present. Figure 6.9 shows a comparison of the product of skews with no MITM and with MITM present.

We can also observe in Figures 6.10 and Figure 6.11 that the RTT variability is slightly higher, and that the prediction accuracy is slightly reduced when comparing with the results obtained when there was no man-in-the-middle attacker present.

These results verify our assumptions that when a man-in-the-middle attacker imposes a constant or near-constant delay it does not modify the skew observed by the nodes, and the product of the skews remains very close 1, leaving the man-

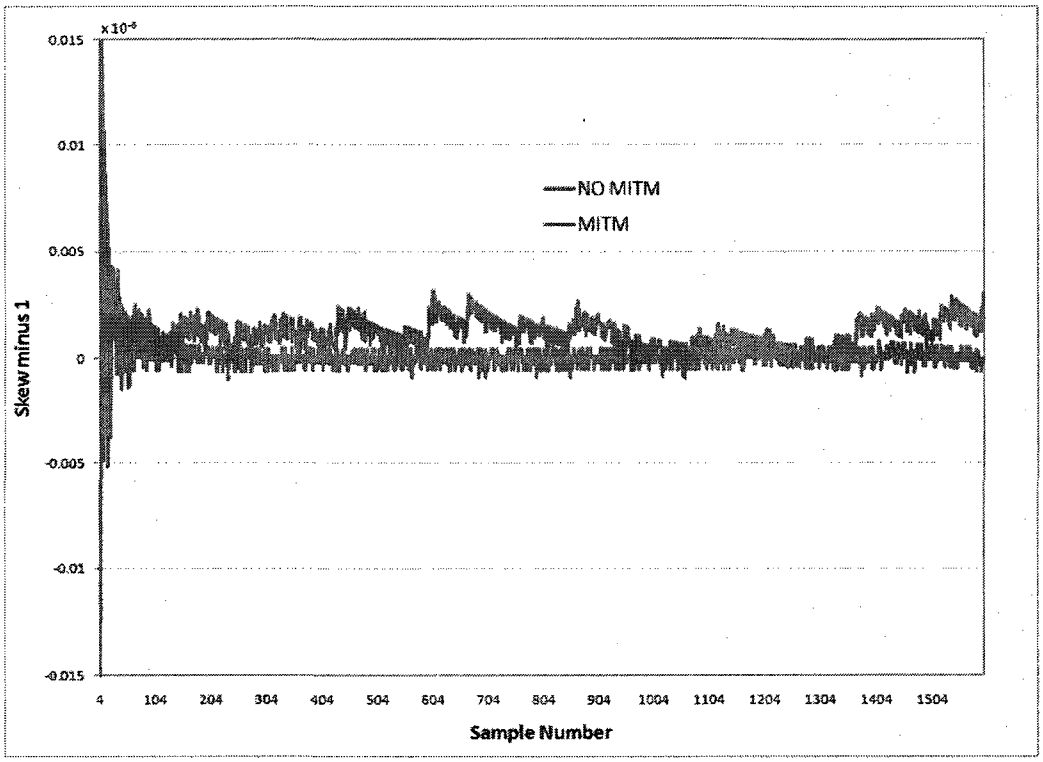


Figure 6.9: Comparison of skews with no MITM and with MITM

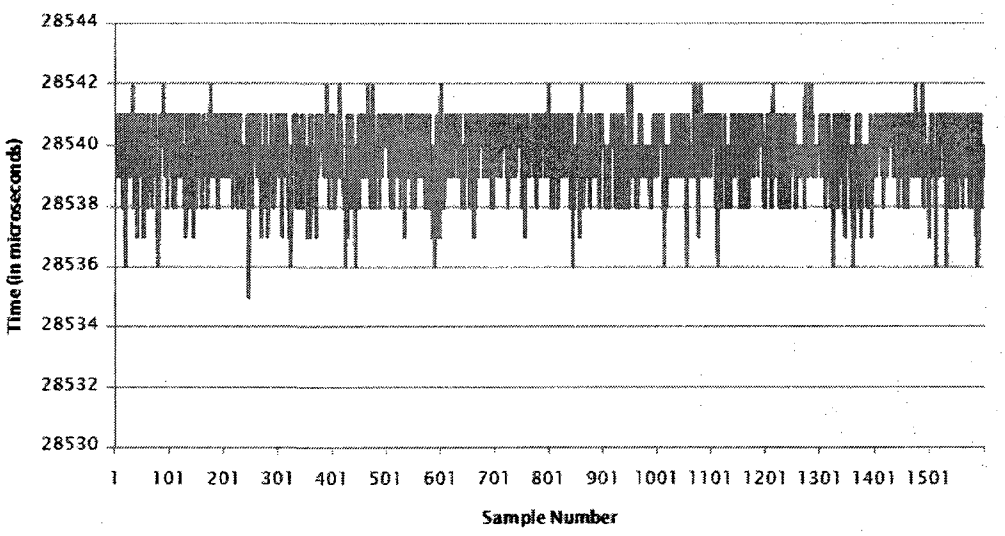


Figure 6.10: Round-trip time observed with MITM present

in-the-middle undetected. The higher variability and reduced accuracy are due to the variable processing delay at the man-in-the-middle attacker, caused by its limited

computation and communication capabilities. Even with this higher variability, the predicted time of arrival is still highly accurate, and therefore, if an attacker linearly increases the delay it induces, it will affect the product of skews and our protocol will be able to detect it.

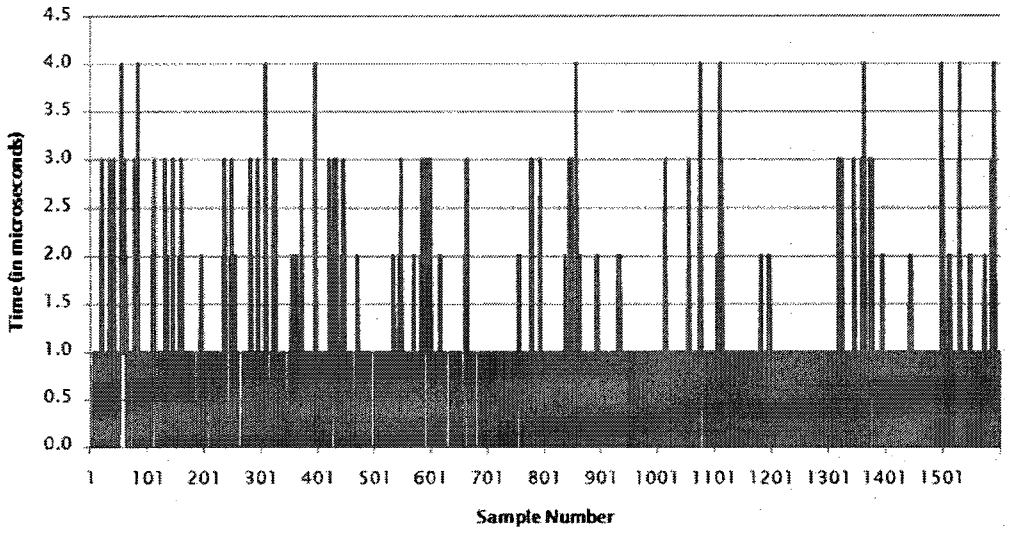


Figure 6.11: Accuracy of packet arrival time prediction with MITM present

Implementation of half-duplex attacker detection

According to Lemma 6.1.2, after having performed the clock synchronization protocol and having measured the round-trip time, the sender needs to transmit a packet to the receiver which is long enough to make the duty cycle of a half-duplex attacker's radio exceed 100%. Due to the CC2420 radio's packet length limit of 127 bytes, it is not possible to send such a long packet. One possibility that first suggests itself is to contiguously transmit several small packets, each of length l with a gap of length g between each successive transmission in order to raise the duty cycle of the link above 50%, which in turn becomes a 100% duty cycle for a half-duplex attacker. We

specifically need to send n packets such that

$$\frac{nl}{nl + (n-1)g + r} > \frac{1}{2},$$

which means that

$$n > \frac{r-g}{l-g}. \tag{6.7}$$

Unfortunately, there is an obstacle to pursuing this approach. The very low speed of the serial interface that connects the CC2420 radio to the memory takes more time to load a packet into the radio's buffer than to transmit the packet, making $g > l$.

Thus, we pursue an alternative solution. We load a packet in the radio's buffer, and then we pseudo-randomly choose to change 1-4 bytes in the buffer between every successive retransmission of the same packet, in a manner known to the sender and receiver, but not to any other nodes. On the receiving side, we selectively copy only the bytes that we know will change in each packet, instead of reading the whole packet. In this way, the time between each successive transmission, becomes smaller than the time to transmit a packet, meaning the man-in-the-middle in the attacker will not have enough time to read and relay each packet.

Validation of half-duplex attacker detection

We built the half-duplex attacker detection mechanism just described and validated it by running it with and without the man-in-the-middle attacker. Without the man-in-the-middle attacker, all tested packets arrived at the estimated time or within 3 μs of it. With the man-in-the-middle attacker present, the average RTT reaches 28540 μs, so we send 10 successive packets in our challenge. The attacker is forced to drop some packets because it is unable to send and receive at the same time, and our defense mechanism is able to catch the attacker in each of the over 100 times

that we tested our scheme. This shows that we can consistently catch a half-duplex man-in-the-middle attacker.

6.4 Security Analysis

We now proceed to analyze the security properties of the proposed protocols. We specifically want to make sure that

1. an attacker is not able to modify the skew in a consistent way, and
2. an attacker can impose only a constant delay.

To verify that we achieve these goals in our experimental testbed, we need to take into account the results presented in Section 6.3, and allow for certain tolerances that reflect the behavior of the system when there is no attacker. By setting these tolerances, we prevent the man-in-the-middle attacker from making changes such as

1. consistently modifying the skew by more than one part in 10^8,
2. changing the one-way delay by more than 6 μs from one packet to the next

6.4.1 Consistent skew

If we set the tolerance of the product of skews to be $|S_{AB}S_{BA} - 1| \leq \varepsilon$, with $\varepsilon > 0$; then the man-in-the-middle attacker will not be able to consistently modify the skew by more than 2ε without detection. Suppose an attacker wants to decrease S'_{AB} (the perceived skew of **B** with respect to **A**). Then the attacker needs to make node **A** think that node **B**'s clock is running slower that it actually is. This can be achieved by *linearly decreasing* the delay m_{AB} it induces. But, from the proof of Theorem 6.1.1, the attacker would be caught after a finite time due to a causality violation. Therefore, the attacker is not able to consistently make $S'_{AB} < S_{AB}$. A similar

argument can be made to demonstrate that an attacker will not able to consistently make $S'_{BA} < S_{BA}$.

Suppose now that the attacker wants to increase S'_{AB}. It needs then to *linearly increase* the delay m_{AB} it induces. But the increase in S'_{AB} results in an increase of the product of skews, and given that the system has established the environmental restriction $|S_{AB}S_{BA} - 1| \leq \varepsilon$, the attacker needs to make sure that $|S'_{AB}S'_{BA} - 1| \leq \varepsilon$ to avoid detection. Then,

$$\begin{aligned} S_{AB}S_{BA} &\geq 1-\varepsilon \\ S'_{AB}S'_{BA} &\leq 1+\varepsilon\,, \end{aligned}$$

resulting in

$$S'_{AB} \leq S_{AB}\frac{1+\varepsilon}{1-\varepsilon}\,,$$

and therefore

$$S'_{AB} \leq S_{AB}(1+2\varepsilon).$$

Furthermore, if the man-in-the-middle attacker does not simultaneously decrease the latency in the reverse direction, the resulting RTT variation will allow the detection of the attacker.

6.4.2 Consistent one-way delay

Empirically, by the use of the formula for calculating the estimated arrival time, as given by (6.6), we can obtain a bound on the tolerable change in perceived one-way delay from one packet to the next. In particular, we can see from Figure 6.5, that when there is no man-in-the-middle attacker, $|\hat{\rho}_{AB}(n) - \rho_{AB}(n)| \leq 2$ μs. Let $\rho_{AB}(n)$ denote the receive time when there is no attacker, and $\rho'_{AB}(n)$ denote the receive time with an attacker present, with $m_{AB}(n) = \rho'_{AB}(n) - \rho_{AB}(n) \geq 0$ by causality, $m_{AB}(n)$

being the delay induced by the attacker.

We have established the tolerance $|\hat{\rho}_{AB}(n) - \rho_{AB}(n)| \leq 2$ μs, and since we will also require that $|\hat{\rho'}_{AB}(n) - \rho'_{AB}(n)| \leq 2$ μs, we can bound

$$|m_{AB}(n) - m_{AB}(n-1)| \leq 4 \text{ } \mu\text{s}\,.$$

Therefore, for an attacker to remain undetected, the delay m_{AB} it induces should not vary more than 4 μs from one packet to the next.

6.4.3 Security of delayed authentication

Let us now evaluate the security of the delayed authentication mentioned in Section 6.2.2.

- All timing information regarding a particular packet is included entirely within that packet. This ensures that if an attacker is not capable of forging or modifying the packet, then the attacker is not able to prevent the receiver from correctly timestamping the packet reception. Furthermore, the attacker cannot make a node choose wrong values for the timing information required (skew, delay) as a result of the delayed authentication.

- Since a session key is never disclosed, forging a delayed authenticator is as hard as forging a non-delayed authenticator. Therefore, the use of delayed authentication does not introduce new authentication attacks.

- Delayed authentication could be vulnerable to Denial-of-Service attacks, where the attacker floods the receiver with spurious packets looking to overflow the receiver's memory and make it discard legitimate packets. However, in the case of the IMote2, there is enough memory to store the spurious packets. This is because the interval for the clock synchronization is 3 seconds and the CC2420

communication rate is 250 kbps, which corresponds to around 92 KB (IMote2 has 256 KB of RAM and 32MB of DRAM) [65].

6.4.4 Security against inter-session replay

For this analysis, we use the following observations:

- It is almost impossible to confuse a packet authenticated using the master key with a packet authenticated with a session key given that both keys are different with a very high probability.
- It is also almost impossible to confuse a pair of packets authenticated with two session keys from different sessions since the session keys are derived using SHA1 and are therefore different with very high probability.
- The only possible confusion is between two packets sent with the master key.
- Only the two first messages sent in the neighbor discovery protocol (INITIATE and RESPOND) are authenticated using the master key.

From the above observations, it is clear we only need to consider the possibility of replay of INITIATE and RESPOND messages. Both of these messages are identified by a type field (to indicate if the message is either an INITIATE or a RESPOND message) and a nonce, which as we mentioned in Section 6.2.3, is a randomly generated number used to ensure the request for a session is fresh.

- If an attacker replays an old INITIATE message to node **B**, and the receiver accepts it as authentic, it will create a RESPOND packet with a fresh nonce η_B, which would result in a different session key that the attacker is not able to generate.

- If an attacker replays an old RESPOND message to node **A**, then node A will find out that the nonce η_A does not correspond to any outstanding INITIATE message it has, ignoring the message.

Therefore, the replay of messages between different sessions is not more likely to succeed than sending a forged message authenticated with a random key.

CHAPTER 7

SECURE NETWORK-WIDE CLOCK SYNCHRONIZATION AND TOPOLOGY DISCOVERY

We now address the problem of secure clock synchronization in a multihop network. Further, we show how the network-wide synchronization can be used for secure topology discovery. In turn, secure topology discovery can be used for network operation, including routing and transport. Our approach to network-wide synchronization will build upon the link synchronization scheme described in Chapter 6.

We use the same notation as in Section 6.1.1, and assume the following:

- The network is modeled as a graph $G = (V, E)$, where V represents the set of nodes in the network, and E represents the links connecting the nodes.
- Clocks are affine.
- Nodes in V can accurately timestamp packets.
- The endpoints of every link in E share a private key, which they use to provide authentication and confidentiality.
- The endpoints of every link in E are legitimate, that is, no node in the set V has been compromised.
- The nodes in V are half-duplex.
- A man-in-the-middle attacker

    1. is a node not included in V, which modifies the characteristics of a link $(a, b) \in E$, in order to disrupt the clock synchronization between a and b,

2. is not able to break the cryptography used in any link in E,

3. can be half-duplex, or full duplex, and

4. is free to do as it pleases, but desires to remain undetected.

- In every well-behaved link $(a,b) \in E$, the product of the skews a and b have with respect to each other, is approximately 1 (i.e., $S_{ab}S_{ba} \approx 1$), and the RTT stays fairly constant. This stems from the results in Section 6.3.2.

The **main properties** of the protocol presented in this chapter are:

Detection The protocol can detect links that misbehave (i.e., compromised links), and tag them.

Dissemination On detection of a compromised link, a node can disseminate an "alert" throughout the network.

Isolation Whenever a node **A** needs to communicate with another node **B**, and the set of paths between them contains one or more paths with misbehaving links, node A can remove the path(s) that contain the misbehaved links after a verification process, effectively isolating them.

This protocol therefore results in a network-wide consistent clock. That is, for any two nodes **A** and **B** in the network, the skews obtained through the k different paths between them, $P_1, P_2, P_3, ..., P_k$, are all approximately equal. If the skews obtained do not obey this property, then we say the skews are inconsistent, which implies that one or more links have been compromised, and we therefore need to remove the incorrectly behaving ones.

7.1 The Main Ideas

To achieve the secure network-wide clock synchronization, each node in the network examines every path it uses to reach the other nodes in search of misbehaving links, i.e., links under the control of man-in-the-middle attackers. When a misbehaving link is found, its usage is discontinued. In order for a node **A** to examine every path to all other nodes in the network, node **A** first learns the topology of the network. The topology learning process, and the detection of misbehaving links, proceed hand-in-hand in an incremental fashion. This process is divided into four steps:

1. Single link checking. Nodes synchronize with their neighbors, and check if the direct link they share is behaving correctly.

2. Neighborhood check. Every node **A** sends a list to its valid neighbors that includes information about each of them.

3. Network consistency check. The valid neighbor lists are processed by each of the nodes in a way that allows them to discover the existence of new nodes, and check for possible misbehaving links in the newly learned paths.

4. Removal of network inconsistencies. When the different paths learned by node **A** to reach node **B** are inconsistent, a test is performed on each of the paths to determine which one(s) must be removed from use.

7.1.1 Single link checking

The goal in this step is for nodes to synchronize with their neighbors and check if the direct link is behaving correctly. This is done by the following procedure:

- Every node in the network synchronizes with each of its neighbors using the protocol described in Section 6.1, with the goal of detecting the existence of misbehaving links.

- To detect the misbehaving links, every pair of nodes **A** and **B** sharing a link, verifies that $S_{AB}S_{BA} \approx 1$. Otherwise, they conclude that an attacker has taken control of the link and its changing its behavior.

With this step, we are guaranteeing the detection of misbehaving links.

7.1.2 Neighborhood check

In the second step, we use the information collected in the first step to isolate the misbehaving links through the following procedure:

- Each of the nodes in the network forms a list of *valid neighbors*. Node **A** considers node **B** a valid neighbor if $S_{AB}S_{BA} \approx 1$.
- The valid neighbor list will only be sent to each of the valid neighbors, and it will be encrypted using the private key shared with such valid neighbor.
- The neighbor list includes the following information for each of the valid neighbors: neighbor ID, skew, and one-way delay.

By not including information regarding the compromised links in the valid neighbor lists, we prevent their use not only by the nodes that detected them, but by any other node in the network since they will not be aware of the existence of such links. We are, therefore, guaranteeing the *isolation* of the compromised links detected in this step.

7.1.3 Network consistency check

The goal in this step is for nodes to discover the existence of nodes not directly connected to them, as well as checking the learned paths for possible misbehaving links. This is done by the following procedure:

- Each node **A** processes the lists of valid neighbors it has received, and updates a table that includes the information for the nodes reachable from **A**, henceforth referred to as reachable nodes table. This table contains the following information for each node which node **A** can reach, either directly or by multihop: node ID, validity flag, aggregate skew, aggregate one-way delay, distance (number of hops), and path (sequence of intermediate hops).

- Updating the reachable nodes table involves calculating the aggregate skew, and aggregate one-way delay for nodes included in the lists received, for which there is no information in the reachable nodes table. It also involves verifying the consistency of the end-to-end skews for nodes for which there is already information in the reachable nodes table. The latter needs to be done in case a different aggregate skew for a node is calculated over a path that is different to the one previously discovered. For instance, if node **A** previously found a path P_1 to node **B** through node **X** with an aggregate skew $\hat{S}_{AB,P_1} = \hat{S}_{AX}\hat{S}_{XB}$, and now finds another path P_2 to node **B** through node **Y** with an aggregate skew $\hat{S}_{AB,P_2} = \hat{S}_{AY}\hat{S}_{YB}$, then P_1 and P_2 are considered consistent if $\hat{S}_{AB,P_1} \approx \hat{S}_{AB,P_2}$ within an acceptable tolerance.

- After processing the lists of valid neighbors received, every node has information in its reachable nodes table on how to reach its direct neighbors and their two-hop neighbors. In order for the nodes to discover their three-hop neighbors, the information in the reachable nodes tables must be exchanged between neighbors. For this, every node copies the contents of its reachable nodes table to a reachable list, and sends it to its valid neighbors (in a similar fashion as the valid neighbor list exchange of step 2)

- The processing of the reachable lists received, similar to the processing of the valid neighbors lists, allow the nodes to discover nodes one hop farther. If every

node sends a reachable list to its neighbors every time there is a change in its reachable nodes table, i.e., a new node or an inconsistency is discovered as a result of the processing of the reachable lists received, eventually every node will discover the network topology. It should be noted that we are actually carrying out a procedure similar to the one performed by the distributed Bellman-Ford algorithm [43], only that instead of computing a minimum cost path, we are aggregating the skew and one way-delay.

With this step, we are guaranteeing the detection of inconsistent skews, and the dissemination of information regarding such inconsistencies throughout the network.

7.1.4 Removal of network inconsistencies

The goal of the last step is for nodes to remove any inconsistency in the set of paths connecting them. This is carried out by the following procedure:

- This step is executed by a node **A** *only* when it needs to communicate with another node **B**. Our approach is an on-demand approach. There is no need for node **A** to pro-actively execute this step for each node it can reach.

- This step allows for the removal of inconsistencies in the set of paths connecting **A** and **B**, by testing each path. The testing process involves sending a packet along the nodes in the path to be tested, and determining if such packet arrived at the estimated time of arrival, which is calculated using (6.3). If the packet does not arrive by the estimated time of arrival (within a certain tolerance), then the path is deemed inconsistent, and is removed from the reachable nodes table.

- This step uses public key signatures to provide authentication and integrity, in order assure nodes as to the origin and content of the messages received [102].

With this step, we are preventing the use of any path that include compromised links.

7.1.5 An example

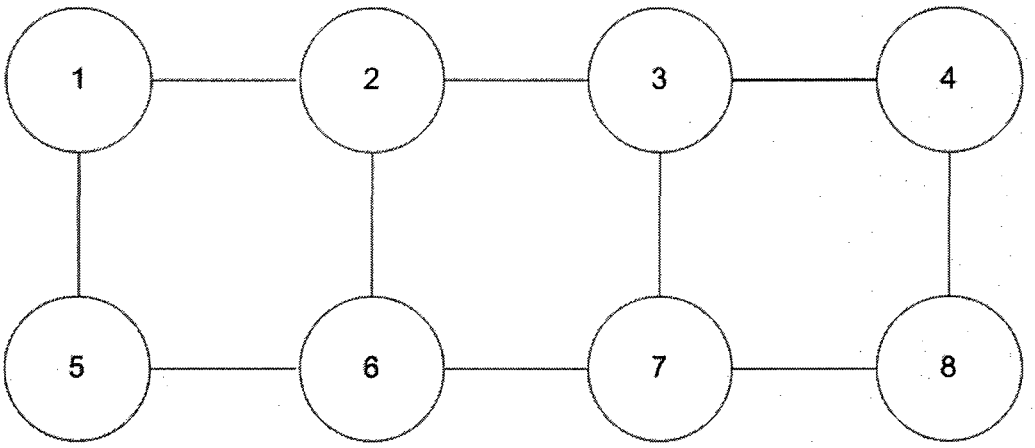


Figure 7.1: Example network to illustrate secure multihop protocol execution

We now illustrate the way in which the nodes discover other nodes outside their neighborhood, and the way in which they handle the possible skew inconsistencies, by using the example network shown in Figure 7.1. Each vertex represents a node in the network, and each edge represents a link which effectively makes them neighbors (i.e., node 7 has nodes 3, 6 and 8 as neighbors, while node 1 has nodes 2 and 5 as its neighbors). All of the nodes in the network synchronize with their neighbors. After that, they each check the links to their neighbors to detect misbehaving links.

Assume nodes 5 and 3 have already synchronized with their neighbors, and that node 3 has detected that the link it shares with node 4 is misbehaving ($\hat{S}_{3,4}\hat{S}_{4,3} \neq 1$). As shown in Figure 7.2, node 5 would then exchange the following list with its neighbors, nodes 1 and 6

$$ListOfNeighbors_5 = \{(1, \hat{S}_{5,1}, \hat{\delta}_{5,1}), (6, \hat{S}_{5,6}, \hat{\delta}_{6,7})\},$$

Algorithm 1 Securely synchronize clocks in a multihop network

Require: Node's clocks are trustworthy

Ensure: Nodes have a network-wise consistent clock

Step 1: Single link check (between all neighboring nodes A and B)

A performs clock synchronization with **B**.

Node **A** obtains the skew $\hat{S}_{AB}$ and the one-way delay $\hat{\delta}_{AB}$.

Node **B** obtains $\hat{S}_{BA}$ and $\hat{\delta}_{BA}$.

Both nodes check that $\hat{S}_{AB} * \hat{S}_{BA} \approx 1$.

Step 2: Neighborhood check

Each of the nodes in the network verifies the validity of the links shared with its neighbors, and forms a valid neighbor list which is to be sent to all valid neighbors. The list includes the measured skew and one-way delay for each of the valid neighbors.

Step 3: Network consistency check

Once it has been determined that a link is valid and can be used, *currentNode* sends a list to the *currentValidNeighbor*. The list contains all reachable nodes that have valid skews, where valid skew means:

1. if a node is a direct neighbor, then an inverse skew relationship holds.
2. if a node is multihop reachable, then the aggregate skews from different paths are consistent.

This step is run until all paths have been explored, that is, all valid reachable nodes have been discovered. If a node is reachable from node **A**, but does not have valid aggregate skews, node **A** tags the node as invalid, and propagate the tags out to inform the other nodes of the inconsistency.

Step 4: Eliminate network skew inconsistencies

If a node **A** needs to communicate with a node **B** that is not a direct neighbor, and node **A** has detected inconsistent skews in the set of paths between **A** and **B**, node **A** then tests each of the inconsistent paths by sending a signed message to node **B**.

The reception timestamps of the messages sent by **A** are returned by **B** and are used to discard paths that do not yield the true skew. This is determined by comparing the estimated time of arrival with the actual time of arrival of the messages sent.

while node 3 would exchange the following list with nodes 2 and 7,

$$ListOfNeighbors_3 = \{(2, \hat{S}_{3,2}, \hat{\delta}_{3,2}), (7, \hat{S}_{3,7}, \hat{\delta}_{3,7})\}.$$

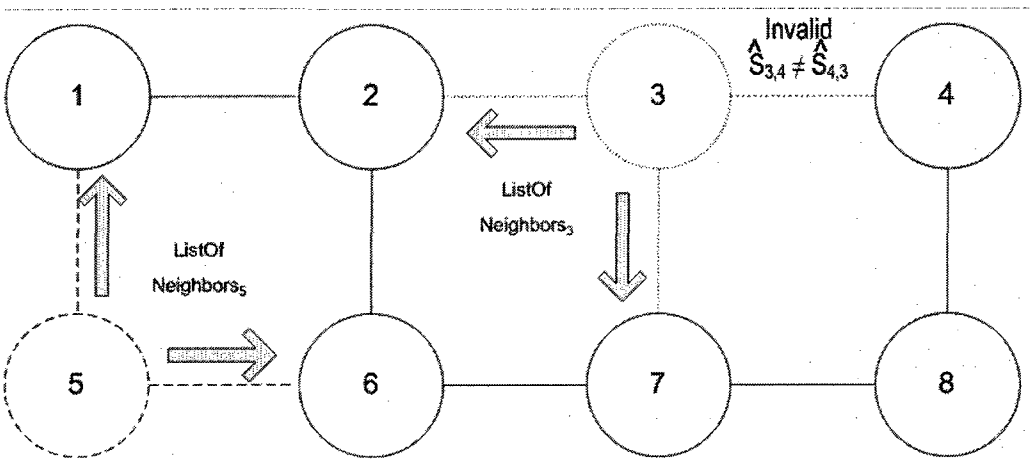


Figure 7.2: Neighborhood check step in nodes 3 and 5

To simplify the illustration we will only describe the events in node 7 for the remainder of the example, noting that similar operations are executed for each node in the network.

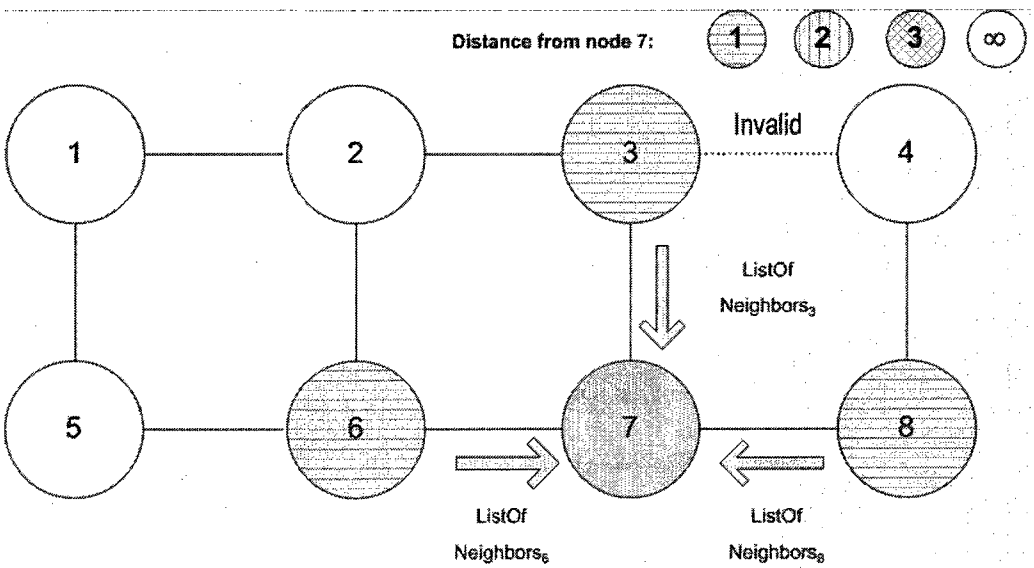


Figure 7.3: Lists of neighbors received in node 7

The different shades in the nodes appearing in Figures 7.3-7.7 denote the distance they have with respect to the node of interest. If a node is not shaded, then node 7 is still not aware of its existence yet. In Figure 7.3 we observe that node 7 receives the neighbor list from its neighbors, nodes 3, 6, and 8. Node 7 proceeds to update its reachable nodes table with the information contained in these lists; for instance, the processing of the list received from node 3 results in the following operations:

- Node 3, which sent $ListOfNeighbors_3$ is already in the reachable nodes table since it is a direct neighbor of node 7, with a distance of 1, a valid flag, $\hat{S}_{7,3}$ as skew, and $\hat{\delta}_{7,3}$ as one-way delay

- Node 2, included in $ListOfNeighbors_3$ is not in the reachable nodes table. Node 7 calculates its distance, which is calculated to be 2, since it comes from a neighbor list (1 hop to node 3 plus another hop to node 2), the aggregate skew as $\hat{S}_{7,2} = \hat{S}_{7,3}\hat{S}_{3,2}$, and the aggregate one-way delay as $\hat{\delta}_{7,2} = \hat{\delta}_{7,3} + \hat{\delta}_{3,2}$. This information, along with (3) as the sequence of intermediate hops, is stored in the reachable nodes table and it is marked as valid since this is the first aggregate skew node 7 calculates with respect to node 2.

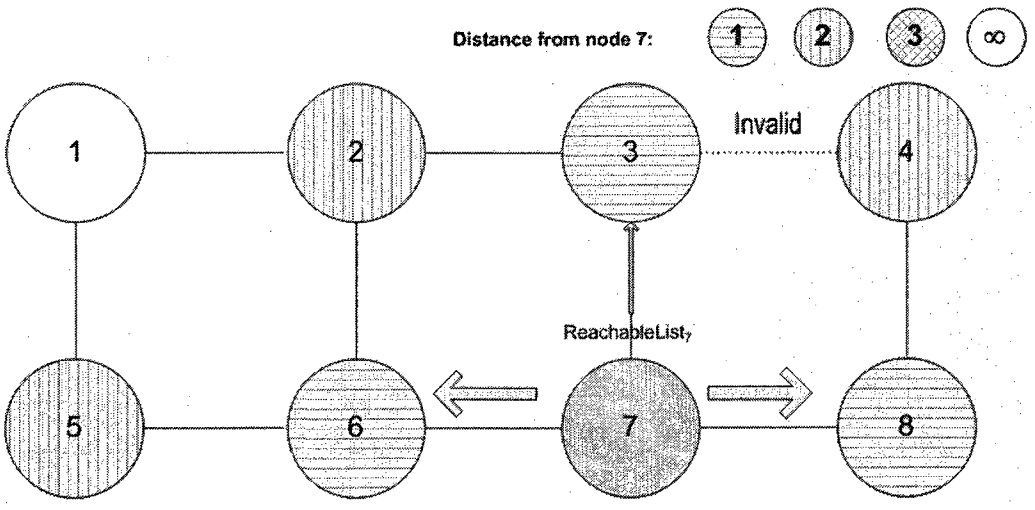


Figure 7.4: Node 7 exchanges its reachable list with its neighbors

When all the lists of neighbors received in node 7 are processed, node 7 is able to reach nodes 3,6,8,4,2, and 5. The fact that the processing of these lists resulted in changes in the reachable nodes table of node 7, means that now node 7 needs to inform the neighboring nodes of such changes, as can be seen in Figure 7.4. For this, node 7 copies the contents of its reachable nodes table into the list $ReachableList_7$.

To illustrate the content of the reachable list, we show the $ReachableList_7$ being sent in Figure 7.4 (the fields for each of the nodes included are node ID, validity flag, distance, skew, one-way delay, and path):

$$\begin{aligned} ReachableList_7 = \{ \quad & (3, valid, 1, \hat{S}_{7,3}, \hat{\delta}_{7,3}, ()), \\ (6, valid, 1, \hat{S}_{7,6}, \hat{\delta}_{7,6}, ()), \quad & (8, valid, 1, \hat{S}_{7,8}, \hat{\delta}_{7,8}, ()), \\ (2, valid, 2, \hat{S}_{7,2}, \hat{\delta}_{7,2}, (3)), \quad & (4, valid, 2, \hat{S}_{7,4}, \hat{\delta}_{7,4}, (8)), \\ (5, valid, 2, \hat{S}_{7,5}, \hat{\delta}_{7,5}, (6))\} & \end{aligned}$$

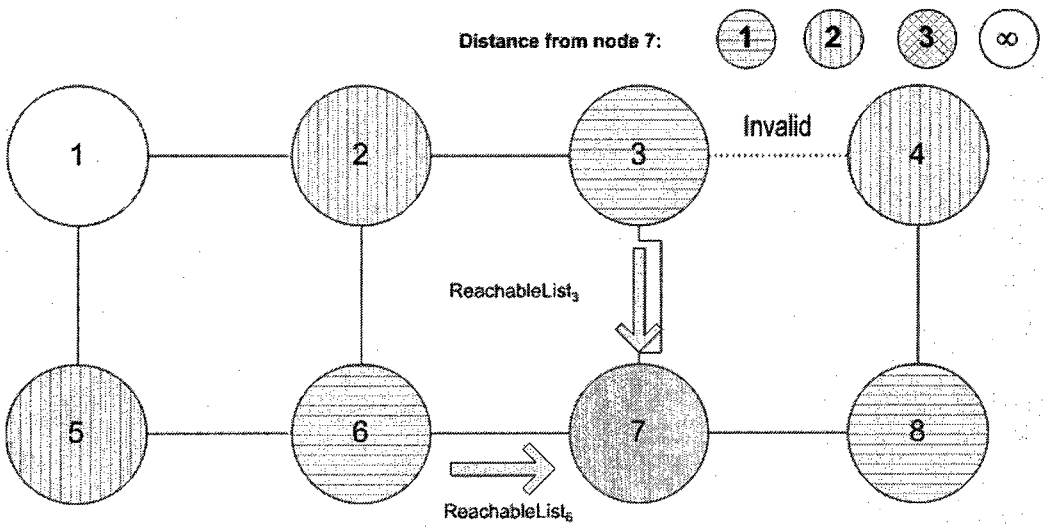


Figure 7.5: Node 7 receives reachable lists from its neighbors

All the other nodes conduct a similar processing of the neighbors lists they receive, which results in updates to their reachable nodes tables. For instance, nodes 3 and 6 become aware of new nodes they can reach, and therefore form a reachable list that

is sent to node 7 (and their other neighbors), as can be seen in Figure 7.5.

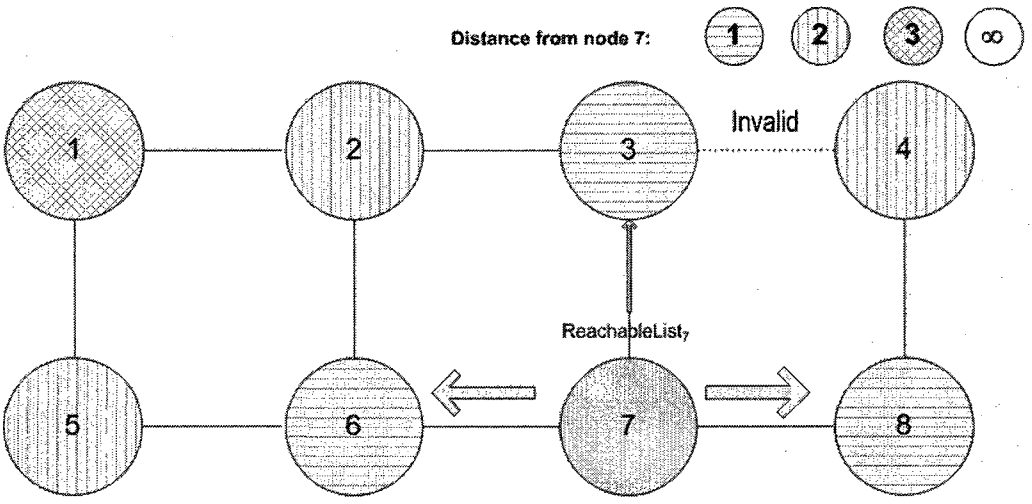


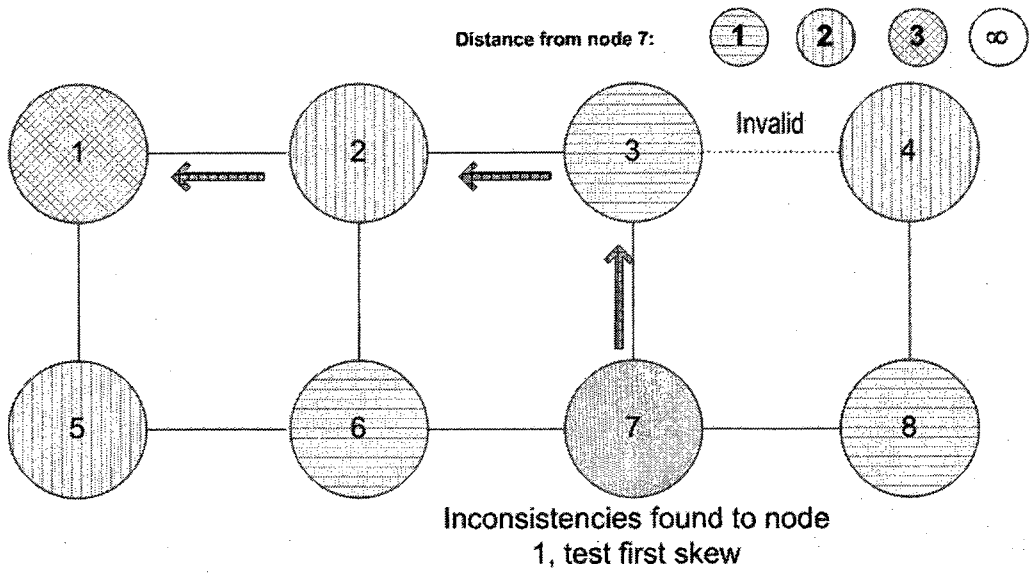
Figure 7.6: Node 7 exchanges its reachable list with neighbors again

The arrival of a reachable list in node 7 triggers an update of the reachable nodes table, as do the neighbor lists. In this case, the update results in the addition of node 1, which previously was not reachable from node 7, which in turn results in the need for node 7 to send another reachable list to its neighbors, as shown in Figure 7.6

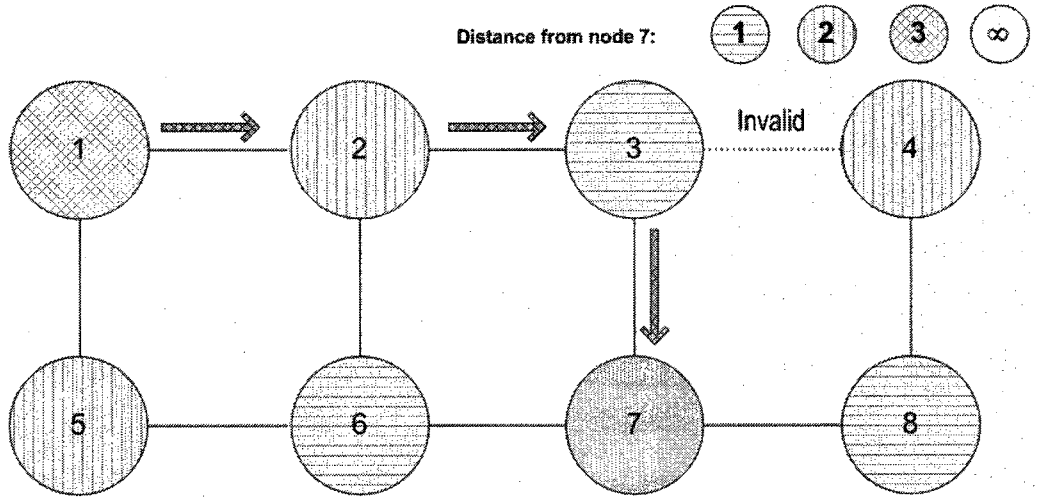
Let us assume now that the aggregate skews that node 7 has calculated with respect to node 1 are different for the two paths it has learned so far, that is, $S_{7,3}S_{3,2}S_{2,1} \neq S_{7,6}S_{6,5}S_{5,1}$. This means that node 7 needs to store the information for both paths, as well as mark node 1 as invalid in the table. The fact that node 1 has been marked as invalid will be communicated to the other nodes in the network by means of the reachable list that will propagated due to the changes in the reachable nodes table.

If at a later time, node 7 needs to communicate with node 1, then node 7 will first have to remove the inconsistent skews. This removal of inconsistencies will involve the following message exchanges illustrated by Figure 7.7, where **A** $\rightarrow$ **B**: M means **A** sends message **M** to **B**, and $\{\mathrm{M}\}_{K_A}$ means message **M** is signed by node A:

- **7** $\rightarrow$ **3**: resynchronize with 2, and append the message $\{\mathrm{nonce}_1\}_{K_7}$



(a) Signed message sent to node 1 to test for inconsistency in first path



(b) Signed reply that includes the time of reception

Figure 7.7: Node 7 wants to communicate with node 1 but needs to first remove the inconsistencies

- **3 → 2**: resynchronize with 1, and append the message $\{\text{nonce}_1\}_{K_7}$
- **2 → 1**: $\{\text{nonce}_1\}_{K_7}$
- **1 → 2**: $\rho_{7,1}(n)$, $\{\rho_{7,1}(n), \{\text{nonce}_1\}_{K_7}\}_{K_1}$
- **2 → 3**: $\rho_{7,1}(n)$, $\{\rho_{7,1}(n), \{\text{nonce}_1\}_{K_7}\}_{K_1}$
- **3 → 7**: $\{\rho_{7,1}(n), \{\text{nonce}_1\}_{K_7}\}_{K_1}$.

Once this message exchange is complete, node 7 is able to determine if the aggregate skew in this path is valid, by verifying if the message arrived at node 1 at the estimated time of arrival. This estimated arrival time is calculated using(6.3). If the skew along this path is valid, we remove the remaining skews, otherwise we proceed to test the next skew in the table.

Using the protocol described by Algorithm 1, the nodes in a network are able to securely synchronize their clocks by detecting and discontinuing the use of misbehaving links created by man-in-the-middle attackers. At the same time, the protocol allows the nodes to securely learn the topology of the network.

7.2 Implementation

The main goal of the prototype implementation is to verify the properties of Algorithm 1. We note that at this stage we do not address efficiency aspects, which can employ several optimizations.

7.2.1 Setup

The implementation has been performed on a testbed comprised of 25 Crossbow IMote2 sensor nodes, shown in Figure 7.8, using TinyOS 2.1, and the implementation discussed in Section 6.2 as a starting point. Figure 7.8 also shows the presence of 6 additional motes that log every packet they hear for later analysis. The reason for using these additional six motes is to keep the nodes participating in the clock synchronization process from having to handle all the interrupts caused by the packet logging, which could adversely affect the timestamp measurements on the motes.

The link topology used in the experiments is shown in Figure 7.9. This topology was enforced by software (MAC filtering), similar to the implementation discussed in

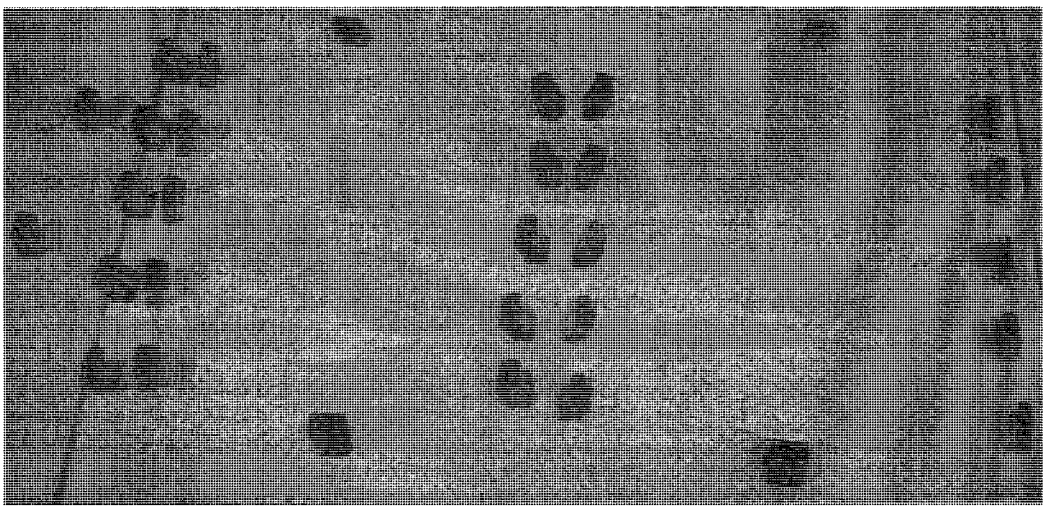


Figure 7.8: Testbed for the implementation of the secure network-wide synchronization protocol

Section 4.2.

We now discuss the implementation of the valid neighbor list and reachable list exchanges, as well as of the process of removing the inconsistent skews.

7.2.2 Exchange of valid neighbor lists

When the clock synchronization between two neighboring nodes **A** and **B** is complete, each of these two nodes has enough information to execute a single link check. This is done by multiplying the estimated skew of one node with respect to the other ($\hat{S}_{AB} * \hat{S}_{BA}$) and checking if the product is approximately 1. As mentioned in Section 6.3.1, a skew is represented by an IEEE 754 double-precision floating point value. Given that there can be variations due to interrupt-handling in the nodes, as well as changes in voltage or temperature that affect the clock drift [103], we allow the product to be within a certain tolerance, which in our case is 10^{-7}.

Every node in the network performs the same verification for all the neighbors it has, and after every verification, the information it has regarding the neighboring node is updated to reflect the validity of the link they share. In order to handle a

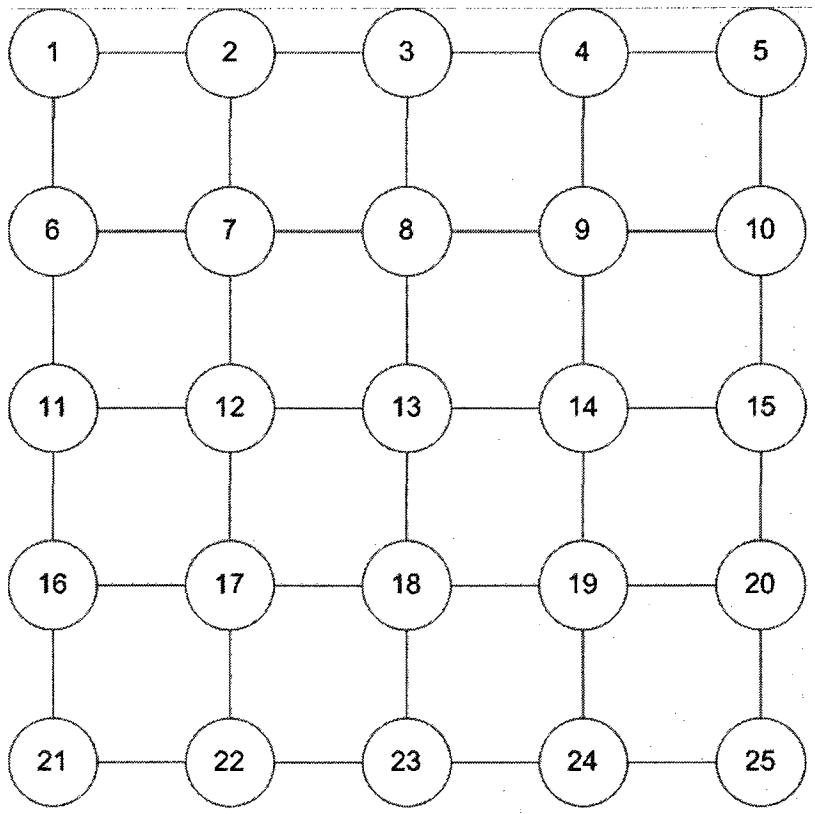


Figure 7.9: Link topology of the 25 nodes in the testbed

dynamic topology, a linked list structure is used to store this information.

Once a node has checked all the direct links it has, it constructs a packet (or set of packets given the CC2420 radio's packet size limit of 127 bytes), that includes the following information for each of its valid neighbors: neighbor ID, validity, distance to the neighbor (1 for direct link neighbors), skew and one-way delay. The packets are encrypted using the session key generated with the protocol described in Section 6.2.3 before sending them to each of the valid neighbors.

7.2.3 Exchange of reachable lists

Whenever node **A** receives a valid neighbor list sent by node **B**, the reachable node table in **A** is updated according to the following rules:

- If the valid neighbor list includes information for a node **C** not currently in **A**'s

table, then **A** calculates the aggregate skew and aggregate one-way delay by combining the corresponding values advertised in the list for node **C**, with the skew and one-way delay that node **A** has with respect to node **B**. The aggregate values are then placed in a new record in the table for node **C**, along with the updated distance, validity and set of intermediate nodes in the path.

- If the valid neighbor list includes information for a node **C** already in **A**'s table, then **A** calculates the aggregate skew and aggregate one-way delay by combining the corresponding values advertised in the list for node **C**, with the skew and one-way delay that node **A** has with respect to node **B**, and compares the computed aggregate skew with the one already stored in the table. If the computed aggregate skew is consistent (e.g., their values are within a tolerance of 10^{-6}), then the information for node **C** remains as valid; otherwise, the computed aggregate values are added to the record for node **C**, along with the distance and set of intermediate nodes in the path, and node **C** is now marked invalid.

Rather than having the nodes sending the reachable lists every time a change in their reachable nodes table is made, which could lead to a high traffic load, the nodes do the following: When a list is received, a timer is started to allow for the arrival of more lists that could trigger additional updates. Once the timer is fired, the node determines whether the reachable nodes table was modified or not by the processing of the received lists. If the table has changed, the node constructs a packet (or set of packets) in the same way it constructed the valid neighbor list, that includes the same information for each node as in the valid neighbor list, with the addition of the set of intermediate nodes that form the path to the node. This list is then encrypted using the corresponding session key and sent to each of the neighbors.

Although in Section 7.1 we referred to the neighbor list and the reachable list as different entities, in the implementation, given that their contents are practically identical, the same data structure is used to represent either one of them. By using the same data structure, the size complexity of the code is reduced.

7.2.4 Removal of network inconsistencies

In the description given in Section 7.1.4, we mentioned that this step uses public key signatures to provide authentication and integrity. Although it has been shown that public key cryptography is feasible on sensor nodes [104–106], the resources required in terms of running time and memory are still to high. For this reason, for the implementation of this part of Algorithm 1, we substituted the public key signature scheme by a delayed authenticator scheme. This latter scheme is the same as the one we used to handle the problem mentioned in Section 6.2.2, of not having enough time to compute an authenticator for the packet being currently sent, and therefore including it in the next packet. By doing this substitution, we are also able to reuse the code already developed to solve that problem.

7.3 Evaluation

We now present the results of the implementation.

Several experiments have been run on the testbed with the following scenarios:

1. No attackers in the network.

2. Attackers in the network. To make the nodes think there is an attacker present in the network, we proceed as follows:

   (a) We make two nodes **A** and **B** sharing a link L advertise incorrect skews to each other, resulting in the immediate isolation of the link since $\hat{S}_{AB} *$

$\hat{S}_{BA} \neq 1$.

(b) We make one of the neighbors of a given node **A** advertise an incorrect skew with respect to **A** to its other neighbors. Whenever any of the nodes in the network uses this incorrect information with respect to **A** to update their reachable nodes table, it will result in inconsistencies that will be propagated throughout the network. These inconsistencies in the reachable nodes tables of the nodes with respect to **A**, will force a node that wants to communicate with node **A** to remove the inconsistencies first.

All the experiments that were run exhibited similar results in all the scenarios mentioned above:

1. The product of the skews between nodes in the network stayed very close to 1, deviating only about one part in 10^7, even on multihop links.

2. The invalid links are either isolated from the beginning (since they are not advertised), or the nodes stop using them as a result of the last step of the protocol (removal of network skew inconsistencies).

In Figures 7.10-7.12, we show the resulting skew estimates of nodes 1, 13, and 22 with respect to all other network nodes (the x-axis represents the ID of the other nodes, referring to Figure 7.9 for the underlying link topology) in a particular run with the following characteristics:

- Nodes 4 and 5 advertised incorrect skews to each other, resulting in the isolation of the link between them. By analyzing the data collected by the loggers, we could verify that the link was not included in the valid neighbor list exchanged by either node, and therefore, that no path in the network included this link.

- Node 6 advertises an incorrect skew with respect to node 7, resulting in skew inconsistencies with respect to node 7 in the reachable nodes tables of all the

other nodes. This was verified by looking at the reachable lists exchanged by the nodes. To verify that the removal of network inconsistencies actually works, node 1 is asked to establish communication with node 7, resulting in the elimination of path $(1, 6, 7)$.

We can see in Figures 7.10-7.12 that the product of the measured skews stays very close to 1, deviating only about one part in 10^7, even with nodes that are not direct neighbors.

To summarize, using the ideas described in Chapter 6 to secure the clock synchronization over a single link as a foundation, we have developed a secure network-wide clock synchronization protocol capable of detecting man-in-the-middle attackers that cannot be prevented by the sole use of cryptographic techniques and that could disrupt the clock synchronization service, or other services such as routing. This protocol also allows us to securely discover the network topology, a functionality that enables us to provide further services such as secure link state routing. The prototype implementation, which has not been optimized for efficiency, has shown the feasibility of the protocol.

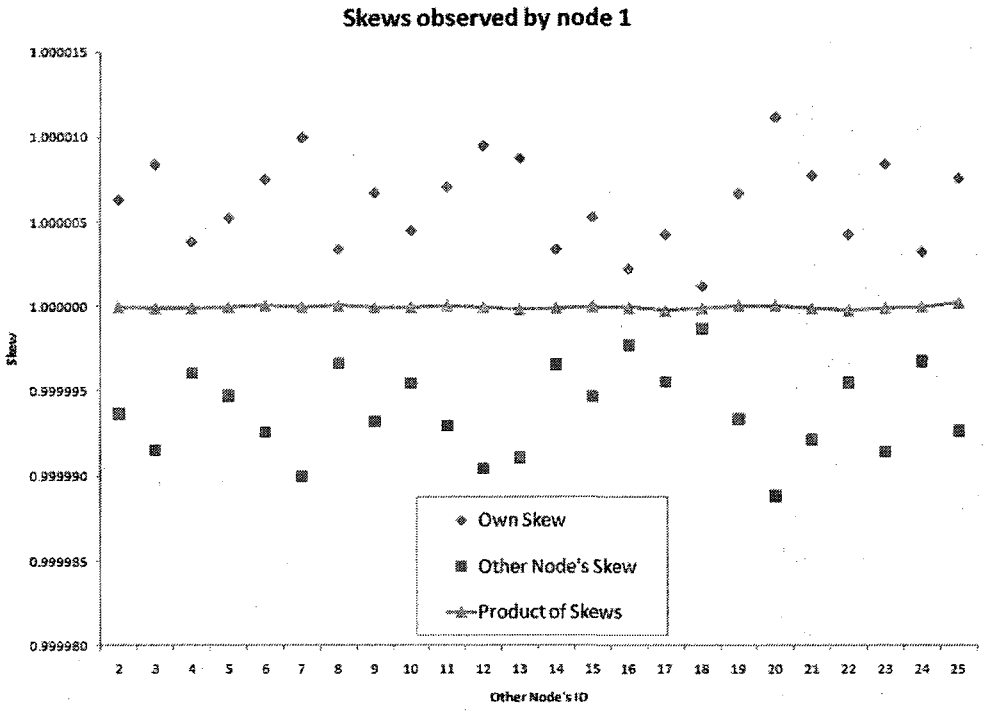


Figure 7.10: Skews observed by node 1 in a particular run of the experiment.

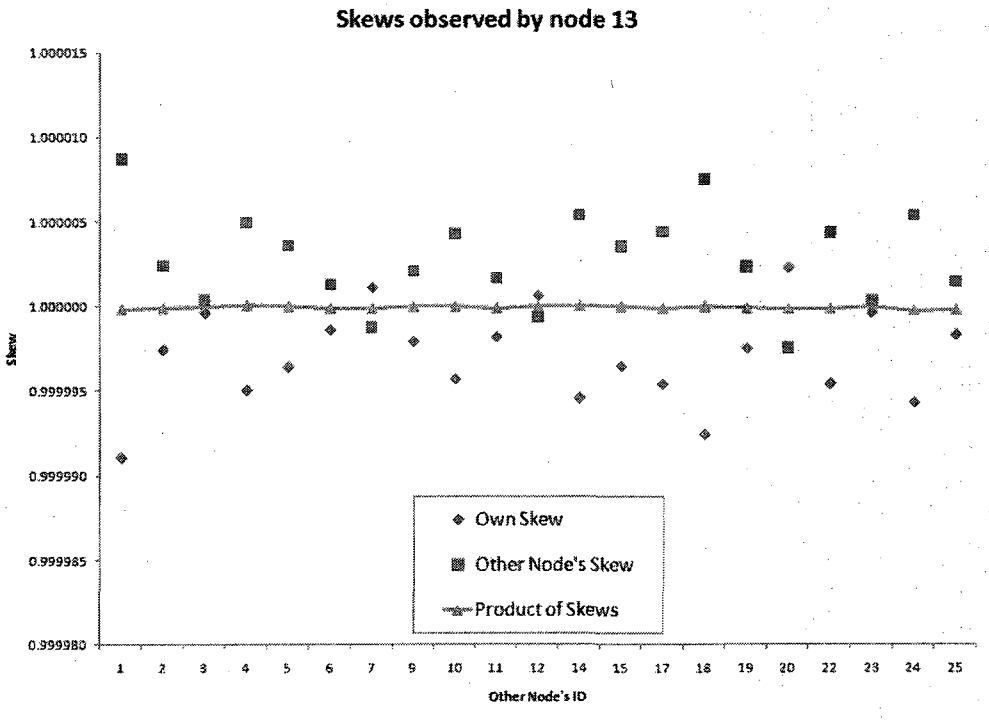


Figure 7.11: Skews observed by node 13 in a particular run of the experiment.

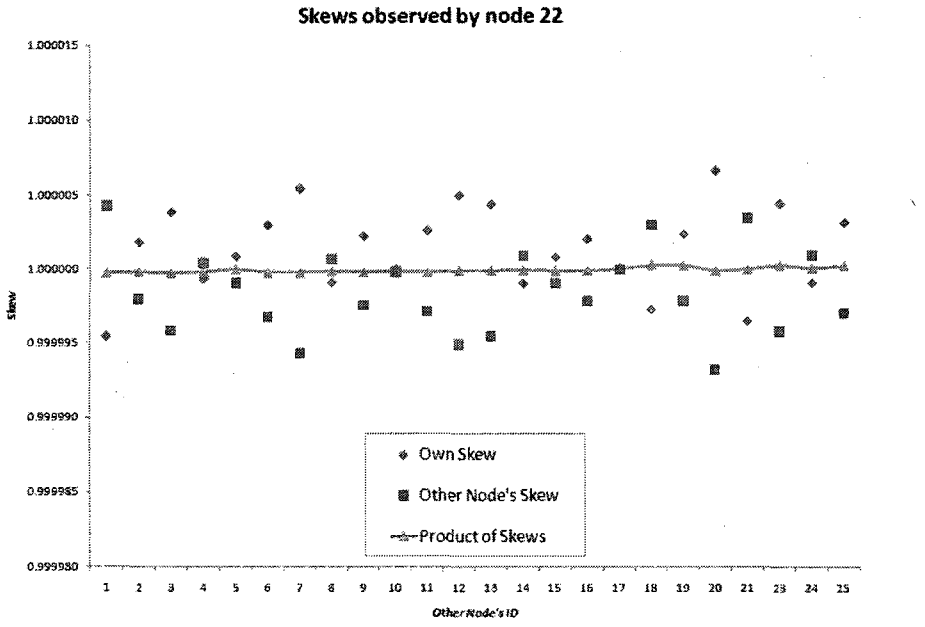


Figure 7.12: Skews observed by node 22 in a particular run of the experiment.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this dissertation we have studied clock synchronization protocols for multihop wireless sensor networks.

In Chapter 4, we presented a clock synchronization protocol tailored to wireless networks, that exploits the broadcast nature of the wireless medium, is fully distributed, uses asynchronous messages, and requires no topological constructions such as rooted trees or hierarchies. These features make the protocol more suitable for dynamic topologies, which can be caused by mobility or node failures, than other protocols such as TPSN [16], where a change in the topology requires running an algorithm to reconstruct the hierarchical structure needed for the synchronization phase.

The protocol has been successfully implemented in TinyOS 1.1 [60] using a testbed of MICA2 motes. Its performance has been evaluated and compared with the leading protocol in use, FTSP [17]. We have shown that our protocol provides a better accuracy and a lower variance. A lower variance may actually be even more important than a good accuracy for some applications such as TDMA communication.

In Chapter 6, we present a secure clock synchronization protocol designed to detect man-in-the-middle attacks. These attacks cannot be prevented by merely using cryptographic techniques, but this protocol is capable of detecting them using timing information alone, with certain conditions. The protocol has been implemented in TinyOS 2.1 using IMote2 motes with very positive results. The evaluation of the implementation shows that a misbehaving link caused by a man-in-the-middle attacker

can be detected if it causes inconsistent skews and variable round-trip delays that result from the non-constant delays induced by the attacker, and that a misbehaving link caused by a half-duplex man-in-the-middle attacker can be detected even when it induces only a constant delay.

In Chapter 7, we presented a secure network-wide clock synchronization protocol. It extends the secure clock synchronization protocol for a single link presented in Chapter 6. The protocol is capable of detecting man-in-the-middle attacks and removing the compromised links from use. The protocol has been implemented in TinyOS 2.1 using a testbed of 25 IMote2 motes, showing the feasibility of the protocol.

8.1 Future Work

A more efficient implementation needs to be built in order to reduce the number of messages exchanged as well as the cryptographic operations involved. A more thorough evaluation of the implementation with different topologies and the presence of a real man-in-the-middle attacker needs to be performed as well as a complete study to determine the number of attackers that the secure clock synchronization protocol could withstand, while still providing a good synchronization. Construction of services such as secure source routing on top of the secure network-wide clock synchronization protocol, is an important next step.

REFERENCES

[1] J.T. Chiang, J.J. Haas, Yih-Chun Hu, P.R. Kumar, and Jihyuk Choi, "Fundamental limits on secure clock synchronization and man-in-the-middle detection in fixed wireless networks," in *INFOCOM 2009. The 28th Conference on Computer Communications. IEEE*, April 2009, pp. 1962–1970.

[2] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton, "Sensor network-based countersniper system," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, New York, NY, USA, 2004, pp. 1–12, ACM.

[3] J. Yick, B. Mukherjee, and D. Ghosal, "Analysis of a prediction-based mobility adaptive tracking algorithm," in *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on*, Oct. 2005, pp. 753–760 Vol. 1.

[4] Tia Gao, D. Greenspan, M. Welsh, R. Juang, and A. Alm, "Vital signs monitoring and patient tracking over a wireless network," in *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, Jan. 2005, pp. 102–105.

[5] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *Internet Computing, IEEE*, vol. 10, no. 2, pp. 18–25, March-April 2006.

[6] Deborah Estrin, Ramesh Govindan, John S. Heidemann, and Satish Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Mobile Computing and Networking*, 1999, pp. 263–270.

[7] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Next century challenges: Mobile networking for "smart dust"," in *International Conference on Mobile Computing and Networking (MOBICOM)*, 1999, pp. 271–278.

[8] Crossbow Technology, "MICA2 Motes," http://www.xbow.com/Products/productsdetails.aspx?sid=174.

[9] Barbara Liskov, "Practical uses of synchronized clocks in distributed systems," in *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, Montreal, Quebec, Canada, 19–21 1991, pp. 1–9.

[10] K. Plarre and P.R. Kumar, "Object tracking by scattered directional sensors," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, Dec. 2005, pp. 3123–3128.

[11] Scott Graham and P. R. Kumar, "Time in general-purpose control systems: The control time protocol and an experimental evaluation," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, dec 2004, pp. 4004–4009.

[12] D.L. Mills, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.

[13] K. Römer, "Time synchronization in ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*. 2001, pp. 173–182, ACM Press.

[14] Jeremy Elson, Lewis Girod, and Deborah Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI-02)*, New York, 9–11 2002, Operating Systems Review, pp. 147–164, ACM Press.

[15] M. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 16–20 2003.

[16] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the first international conference on Embedded networked sensor systems (SenSys-03)*, New York, 5–7 2003, pp. 138–149, ACM Press.

[17] Miklos Maroti, Gyula Simon, Branislav Kusy, and Akos Ledeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, MD, USA, nov 2004, pp. 39–49.

[18] Jana van Greunen and Jan Rabaey, "Lightweight time synchronization for sensor networks," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, apr 2003, pp. 11–19.

[19] S. Palchaudhuri, A. K. Saha, and D. B. Johnson, "Adaptive clock synchronization in sensor networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, apr 2004.

[20] Kyoung lae Noh, E. Serpedin, and K. Qaraqe, "A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization," *Wireless Communications, IEEE Transactions on*, vol. 7, no. 9, pp. 3318–3322, September 2008.

[21] Kyoung-Lae Noh, Yik-Chung Wu, Khalid Qaraqe, and Bruce W. Suter, "Extension of pairwise broadcast clock synchronization for multicluster sensor networks," *EURASIP J. Adv. Signal Process*, vol. 2008, pp. 1–10, 2008.

[22] Philipp Sommer and Roger Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *IPSN '09: Proceedings of the 8th international conference on Information processing in sensor networks*, New York, NY, USA, 2009, ACM.

[23] L. Schenato and G. Gamba, "A distributed consensus protocol for clock synchronization in wireless sensor network," in *Decision and Control, 2007 46th IEEE Conference on*, Dec. 2007, pp. 2289–2294.

[24] Branislav Kusy, Prabal Dutta, Philip Levis, Miklos Maroti, Akos Ledeczi, and David Culler, "Elapsed time on arrival; a simple and versatile primitive for canonical time synchronisation services," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 239–251, 2006.

[25] Gang Xiong and Shalinee Kishore, "Discrete-time second-order distributed consensus time synchronization algorithm for wireless sensor networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2009, pp. 1–12, 2009.

[26] Yingchun Shen and Hai Jin, "Agent-based timing-sync algorithm for sensor networks," *Networks Security, Wireless Communications and Trusted Computing, International Conference on*, vol. 1, pp. 338–344, 2009.

[27] Markus Wlchli, Reto Zurbuchen, Thomas Staub, and Torsten Braun, "Gravity-based local clock synchronization in wireless sensor networks.," in *Networking*, Luigi Fratta, Henning Schulzrinne, Yutaka Takahashi, and Otto Spaniol, Eds. 2009, vol. 5550 of *Lecture Notes in Computer Science*, pp. 907–918, Springer.

[28] Nitthita Chirdchoo, Wee-Seng Soh, and Kee Chaing Chua, "Mu-sync: a time synchronization protocol for underwater mobile networks," in *WuWNeT '08: Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, New York, NY, USA, 2008, pp. 35–42, ACM.

[29] Philipp Blum, Lennart Meier, and Lothar Thiele, "Improved interval-based clock synchronization in sensor networks," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, New York, NY, USA, 2004, pp. 349–358, ACM.

[30] K. Shahzad, A. Ali, and N.D. Gohar, "Etsp: An energy-efficient time synchronization protocol for wireless sensor networks," in *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, March 2008, pp. 971–976.

[31] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA-02)*, New York, 28 2002, pp. 88–97, ACM Press.

[32] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong, "A macroscope in the redwoods," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005, pp. 51–63, ACM.

[33] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke, "Data collection, storage, and retrieval with an underwater sensor network," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005, pp. 154–165, ACM.

[34] Mo Li and Yunhao Liu, "Underground structure monitoring with wireless sensor networks," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 69–78.

[35] Chris R. Baker, Kenneth Armijo, Simon Belka, Merwan Benhabib, Vikas Bhargava, Nathan Burkhart, Artin Der Minassians, Gunes Dervisoglu, Lilia Gutnik, M. Brent Haick, Christine Ho, Mike Koplow, Jennifer Mangold, Stefanie Robinson, Matt Rosa, Miclas Schwartz, Christo Sims, Hanns Stoffregen, Andrew Waterbury, Eli S. Leland, Trevor Pering, and Paul K. Wright, "Wireless sensor networks for home health care," in *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, Washington, DC, USA, 2007, pp. 832–837, IEEE Computer Society.

[36] Hairong Yan, Youzhi Xu, and Mikael Gidlund, "Experimental e-health applications in wireless sensor networks," in *CMC '09: Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, Washington, DC, USA, 2009, pp. 563–567, IEEE Computer Society.

[37] B. Sinopoli, C. Sharp, S. Schaffert, L. Schenato, and S. Sastry, "Distributed control applications within sensor networks," in *IEEE Proceedings Special Issue on Distributed Sensor Networks*, Nov 2003.

[38] Chih-Yu Lin, Wen-Chih Peng, and Yu-Chee Tseng, "Efficient in-network moving object tracking in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, pp. 1044–1056, 2006.

[39] Scott Graham and P. R. Kumar, "The convergence of control, communication and computation," *Proceedings of PWC 2003: Personal Wireless Communication, Lecture Notes in Computer Science*, vol. 2775, pp. 458–475, Jan 2003.

[40] Scott Graham, Girish Baliga, and P. R. Kumar, "Issues in the convergence of control with communication and computing: Proliferation, architecture, design, services, and middleware," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, dec 2004, pp. 1466–1471.

[41] C. Fischione, K. H. Johansson, F. Graziosi, and F. Santucci, "Distributed cooperative processing and control over wireless sensor networks," in *IWCMC '06: Proceedings of the 2006 international conference on Wireless communications and mobile computing*, New York, NY, USA, 2006, pp. 1311–1316, ACM.

[42] R. Rozovsky and P. Kumar, "SEEDEX: A MAC protocol for ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*, 2001, pp. 67–75.

[43] D. Bertsekas and R. Gallager, *Data networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[44] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer, "Dozer: ultra-low power data gathering in sensor networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, New York, NY, USA, 2007, pp. 450–459, ACM.

[45] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Communications of the ACM*, jul 1978, pp. 558–565.

[46] Danny Dolev, Joe Halpern, and H. Raymond Strong, "On the possibility and impossibility of achieving clock synchronization," in *Proceedings of the sixteenth annual ACM Symposium on Theory of Computing, Washington, DC, April 30–May 2, 1984*, ACM, Ed., New York, NY, USA, 1984, pp. 504–511, ACM Press.

[47] Jennifer Lundelius and Nancy Lynch, "An upper and lower bound for clock synchronization," *Information and Control*, vol. 62, no. 2/3, pp. 190–204, / 1984.

[48] Saad Biaz and Jennifer L. Welch, "Closed form bounds for clock synchronization under simple uncertainty assumptions," *IPL: Information Processing Letters*, vol. 80, 2001.

[49] Omer Gurewitz, Israel Cidon, and Moshe Sidi, "One-way delay estimation using network-wide measurements," *IEEE/ACM Trans. Netw.*, vol. 14, no. SI, pp. 2710–2724, 2006.

[50] N.M. Freris and P.R. Kumar, "Fundamental limits on synchronization of affine clocks in networks," *Decision and Control, 2007 46th IEEE Conference on*, pp. 921–926, Dec. 2007.

[51] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler, "Fault-tolerant clock synchronization in distributed systems," *Computer*, vol. 23, no. 10, pp. 33–42, oct 1990.

[52] Leslie Lamport and P. M. Melliar-Smith, "Byzantine clock synchronization," in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, 27–29 1984, pp. 68–74.

[53] J. Lundelius and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," in *3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing.* aug 1984, pp. 75–88, ACM.

[54] Dolev, Halpern, Simons, and Strong, "Dynamic fault-tolerant clock synchronization," *JACM: Journal of the ACM*, vol. 42, 1995.

[55] Leslie Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM*, vol. 32, no. 1, pp. 52–78, jan 1985.

[56] F. Cristian, "A probabilistic approach to distributed clock synchronization," *Distributed Computing*, vol. 3, pp. 146–158, 1989.

[57] K. Arvind, "Probabilistic clock synchronization in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, pp. 474–487, may 1994.

[58] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 933–940, 1987.

[59] Jason Hill, Robert Szewczyk, Alec Woo, Hollar Hollar, David Culler, and Kristofer Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, nov 2000.

[60] "TinyOS Community Forum," http://www.tinyos.net/.

[61] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler, "The nesC language: A holistic approach to networked embedded systems," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, New York, NY, USA, 2003, pp. 1–11, ACM.

[62] Jason Hill and David Culler, "A wireless embedded sensor architecture for system-level optimization," Tech. Rep., UC Berkeley, 2001.

[63] Crossbow Technology, "Wireless Sensor Network Platforms," http://www.xbow.com/Products/wproductsoverview.aspx.

[64] Crossbow Technology, "MICA Motes," http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA.pdf.

[65] Robert Adler, Mick Flanigan, Jonathan Huang, Ralph Kling, Nandakishore Kushalnagar, Lama Nachman, Chieh-Yih Wan, and Mark Yarvis, "Intel Mote 2: an advanced platform for demanding sensor network applications," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005, pp. 298–298, ACM.

[66] The TinyOS 2.x Working Group, "TinyOS 2.0," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005, pp. 320–320, ACM.

[67] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler, "Flexible hardware abstraction for wireless sensor networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, Jan.-2 Feb. 2005, pp. 145–157.

[68] Moteiv, *Tmote Sky Datasheet http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf*, 2006.

[69] P. R. Kumar and P. P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control*, Prentice Hall, Englewood Cliffs, NJ, 1986.

[70] "ECS oscillators, inc," http://www.ecsxtal.com.

[71] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi, "The flooding time synchronization protocol," Tech. Rep., Vanderbilt, 2004.

[72] A. Giridhar and P.R. Kumar, "Distributed clock synchronization over wireless networks: Algorithms and analysis," in *Decision and Control, 2006 45th IEEE Conference on*, Dec. 2006, pp. 4915–4920.

[73] Richard Karp, Jeremy Elson, Deborah Estrin, and Scott Shenker, "Optimal and global time synchronization in sensornets," Tech. Rep., UCLA, 2003.

[74] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar, "Spins: security protocols for sensor networks," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, New York, NY, USA, 2001, pp. 189–199, ACM.

[75] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, Inc., New York, NY, USA, 1995.

[76] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," http://www.ietf.org/rfc/rfc2104.txt, February 1997.

[77] Ross J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, 2 edition, April 2008.

[78] Madhukar Anand, Eric Cronin, Micah Sherr, Matt Blaze, Zachary Ives, and Insup Lee, "Sensor network security: more interesting than you think," in *HOTSEC'06: Proceedings of the 1st USENIX Workshop on Hot Topics in Security*, Berkeley, CA, USA, 2006, pp. 5–5, USENIX Association.

[79] John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary, "Wireless sensor network security: A survey, in book chapter of security," in *in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds.* 2007, pp. 0–849, CRC Press.

[80] A.D. Wood and J.A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, Oct 2002.

[81] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig, "The sybil attack in sensor networks: analysis & defenses," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, New York, NY, USA, 2004, pp. 259–268, ACM.

[82] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *Security and Privacy, 2005 IEEE Symposium on*, May 2005, pp. 49–63.

[83] Haowen Chan and A. Perrig, "Security and privacy in sensor networks," *Computer*, vol. 36, no. 10, pp. 103–105, Oct. 2003.

[84] Y.-C. Hu, A. Perrig, and D.B. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, March-3 April 2003, vol. 3, pp. 1976–1986 vol.3.

[85] X. Wang, Sriram Chellappan, W. Gu, W. Yu, and D. Xuan, "Search-based physical attacks in sensor networks," in *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, 2005, pp. 489–496.

[86] Xun Wang, Wenjun Gu, Kurt Schosek, Sriram Chellappan, and Dong Xuan, "Sensor network configuration under physical attacks," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 4, no. 3/4, pp. 174–182, 2009.

[87] Chris Karlof, Naveen Sastry, and David Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, New York, NY, USA, 2004, pp. 162–175, ACM.

[88] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia, "LEAP: efficient security mechanisms for large-scale distributed sensor networks," in *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, New York, NY, USA, 2003, pp. 62–72, ACM.

[89] Haowen Chan and A. Perrig, "Pike: peer intermediaries for key establishment in sensor networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, March 2005, vol. 1, pp. 524–535 vol. 1.

[90] Laurent Eschenauer and Virgil D. Gligor, "A key-management scheme for distributed sensor networks," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, New York, NY, USA, 2002, pp. 41–47, ACM.

[91] J. Eriksson, S.V. Krishnamurthy, and M. Faloutsos, "Truelink: A practical countermeasure to the wormhole attack in wireless networks," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, Nov. 2006, pp. 75–84.

[92] Michael Manzo, Tanya Roosta, and Shankar Sastry, "Time synchronization attacks in sensor networks," in *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, New York, NY, USA, 2005, pp. 107–116, ACM.

[93] A. Boukerche and D. Turgut, "Secure time synchronization protocols for wireless sensor networks," *Wireless Communications, IEEE*, vol. 14, no. 5, pp. 64–69, October 2007.

[94] Saurabh Ganeriwal, Srdjan Čapkun, Chih-Chieh Han, and Mani B. Srivastava, "Secure time synchronization service for sensor networks," in *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, New York, NY, USA, 2005, pp. 97–106, ACM.

[95] Kun Sun, Peng Ning, and Cliff Wang, "Secure and resilient clock synchronization in wireless sensor networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 2, pp. 395–408, Feb. 2006.

[96] Kun Sun, Peng Ning, and Cliff Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, New York, NY, USA, 2006, pp. 264–277, ACM.

[97] Yang Xiao, Venkata Krishna Rayi, Bo Sun, Xiaojiang Du, Fei Hu, and Michael Galloway, "A survey of key management schemes in wireless sensor networks," *Comput. Commun.*, vol. 30, no. 11-12, pp. 2314–2341, 2007.

[98] "Chipcon CC2420 Datasheet: http://focus.ti.com/lit/ds/symlink/cc2420.pdf, Texas Instruments," 2007.

[99] Miklos Marotti and Janos Sallai, "Packet timestamping," TinyOS Extension Proposal : 132 , may 2008.

[100] "Specification for the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, 2001.

[101] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," http://www.ietf.org/rfc/rfc3174.txt, September 2001.

[102] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, Nov 1976.

[103] John R. Vig, "Introduction to quartz frequency standards," Tech. Rep., Army Research Laboratory, Electronics and Power Sources Directorate, October 1992.

[104] Gunnar Gaubatz, Jens-Peter Kaps, Erdinc Ozturk, and Berk Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," in *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, Washington, DC, USA, 2005, pp. 146–150, IEEE Computer Society.

[105] D.J. Malan, M. Welsh, and M.D. Smith, "A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography," in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, Oct. 2004, pp. 71–80.

[106] An Liu and Peng Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, Washington, DC, USA, 2008, pp. 245–256, IEEE Computer Society.

AUTHOR'S BIOGRAPHY

Roberto Solís Robles was born in Zacatecas, México, on July 21, 1971. He graduated from Instituto Tecnológico de Zacatecas, México, in 1992 with a degree in Ingeniería en Sistemas Computacionales. He has worked in such institution as a professor since February 16, 1993, and also in Universidad Autónoma de Zacatecas, where he lectures courses on Operating Systems, Compilers, Networking, and Object Oriented Programming. He completed a graduate degree of Maestría en Ciencias de la Computación from Instituto Tecnológico y de Estudios Superiores de Monterrey in 1999 and, in 2000, he relocated to Urbana, Illinois to pursue a Ph.D. degree in computer science having been awarded a Fulbright scholarship.