ABSTRACT

SUN, KUN. Trustworthy and Resilient Time Synchronization in Wireless Sensor Networks. (Under the direction of Dr. Peng Ning and Dr. Cliff Wang.)

Wireless sensor networks have received a lot of attention recently due to its wide applications. Accurate and synchronized time is crucial in many sensor network applications. A number of time synchronization schemes have been proposed recently to address the resource constraints in sensor networks. However, all these techniques cannot survive malicious attacks in hostile environments.

This dissertation includes three techniques to achieve secure time synchronization in different scopes of sensor networks. First, we develop a secure single-hop pair-wise time synchronization technique that provides time difference between two neighbor nodes using hardware-assisted, authenticated medium access control (MAC) layer timestamping. This technique can effectively defeat external attacks that attempt to mislead single-hop pair-wise time synchronization.

Second, we propose a fault-tolerant cluster-wise time synchronization scheme to provide a common clock among a cluster of nodes, where the nodes in the cluster can communicate through broadcast. This scheme guarantees an upper bound of time difference between normal nodes in a cluster, provided that the malicious nodes are no more than one third of the cluster. Unlike the traditional fault-tolerant time synchronization approaches, the proposed technique does not introduce collisions between synchronization messages, nor does it require costly digital signatures.

Third, we propose two secure and resilient global time synchronization schemes: *level-based time synchronization* and *diffusion-based time synchronization*. The basic idea of both schemes is to provide redundant ways for one node to synchronize its clock with another far-away node, so that it can tolerate partially missing or false synchronization information provided by compromised nodes. The level-based scheme builds a level hierarchy in the sensor network, and then synchronizes the whole network level by level. The diffusion-based scheme allows each node to diffuse its clock to its neighbor nodes after it has been synchronized. Both schemes are secure against external attacks and resilient against compromised nodes. We implement a secure and resilient global time synchronization protocol, *TinySeRSync*, on MICAz motes running TinyOS. The experimental results indicate that TinySeRSync is a practical system for secure and resilient global time synchronization in wireless sensor networks.

Trustworthy and Resilient Time Synchronization in Wireless Sensor Networks

by

Kun Sun

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh

2006

Approved By:

To my wife Jing, my parents and brother for their endless love.

Biography

Kun Sun received his B.S. degree and M.E. degree in Computer Science from Nankai University, Tianjin, China, in 1997 and 2000. From 2000 to 2001, he worked as a Member of Technical Staff in Bell Labs Asia Pacific and China, Lucent Technology, China. In August 2001, he started his Ph.D. study at North Carolina State University in the Department of Computer Science. His research focuses on developing systems and techniques to enhance security of networks and distributed systems, especially the security of wireless ad-hoc and sensor networks.

Acknowledgements

This endeavor was truly a learning experience. I would like to thank all those who have supported and encouraged me during my five years Ph.D. study.

First, I would like to thank my adviser, Dr. Peng Ning, and my co-adviser, Dr. Cliff Wang. I have benefited so much from their insight, wisdom, suggestion, and constructive criticism of my work. I owe them a lot for their guidance, patience, encouragement, and financial support. I would like to thank my committee members, Dr. Douglas S. Reeves, Dr. Mladen A. Vouk, and Dr. Wenye Wang, for their valuable feedback and comments on my research.

Second, I would like to thank National Science Foundation (NSF) and Army Research Office (ARO) for their funding support.

Third, I would like to give my thanks to all my friends for their help in my Ph.D study: Fang Feng, Yiquan Hu, An Liu, Donggang Liu, Pai Peng, Julia M. Star, Pan Wang, Dingbang Xu, Yan Zhai, Qing Zhang, Qinghua Zhang, Yi Zhang, and Yuzheng Zhou.

Finally, I would like to give my special thanks to my wife Jing, my parents and my brother for their endless encouragement and support during my Ph.D. study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks have received a lot of attention recently due to their wide applications, such as target tracking, monitoring of critical infrastructures, and scientific exploration in dangerous environments. Wireless sensor networks typically consist of a large number of small sensor nodes and possibly a few powerful control nodes. The sensor nodes sense the environmental changes and report them to the control nodes over a flexible network architecture. A sensor node is usually composed of one or a few sensing components, which are able to sense conditions (e.g. light, sound, temperature, pressure) from its surroundings, a processing component, which is able to carry out simple computation, and a communication component, which is capable of short-range wireless communication. The control nodes may further process the data collected from sensor nodes, disseminate control commands to sensor nodes, and connect the network to a traditional wired network. The control nodes can have workstation/laptop class processors, enough memory, energy, and computational power to perform their tasks.

The main challenge of designing wireless sensor networks come from the resource constraint of the sensor nodes. For example, a typical MICAz mote has a 8-bit Atmel ATmega128L processor, 7.3728MHz clock, 4K bytes RAM, 128K bytes of flash for program memory, and 250 kbps radio date rate. Moreover, the MICAz mote is powered by 2 AA batteries, which is difficult to replace when consumed in some hostile environments. Table 1.1 shows the basic characteristics of typical MICAz and MICA2 motes [20], which are widely used in current generation of wireless sensor networks.

Time synchronization is an important component of a wireless sensor network to provide

Table 1.1: Characteristics of MICAz and MICA2 motes.

	MICAz	MICA2
Processor	ATMega128L, 8-bit	ATMega128L, 8-bit
Clock	7.3728 MHz	7.3728 MHz
RAM	4K bytes	4K bytes
Program Flash Memory	128K bytes	128K bytes
RF Transceiver	CC2420	CC1000
RF Transceiver Frequency	2.4 GHz	315-916 MHz
Radio Data Rate	250 kbps	38.4 kbps
Power Supply	2 AA Battery	2 AA Battery
Hardware Security	128 bit AES	No

a common clock in sensor nodes. Most wireless sensor network applications, such as data fusion [60, 103], target tracking [15, 95], and power saving [100], require a synchronized local clocks in sensor nodes. The ability of the wireless sensor network to aggregate the data collected can greatly reduce the number of messages that need to be transmitted across the network. Many data-fusion algorithms [60, 103] have to process the sensor readings ordered by the time of occurrence (e.g., the time when a forest fire was sensed). In target tracking applications [15, 95], sensor nodes need both the location and the time when the target is sensed in order to correctly determine the target moving direction and speed. Several approaches intend to improve the energy efficiency by frequently switching sensor nodes into power-saving sleep mode [100]. A group of nodes needs a common synchronized clock to synchronize their behaviors on switching between wake-up and sleep modes at the same time. The time slotted MAC protocols (e.g., [66]) achieve the multiple access to the shared communication medium by assigning time slots to a group of nodes. Thus, sensor nodes need to have a synchronized clock to access their time slots without colliding with other nodes.

The small sensor nodes usually contain inexpensive oscillators with typical clock drift rates at about tens of microseconds per second [81], and the clock drift (almost 1 second apart per day) is intolerable for the above wireless sensor network applications. Therefore, time synchronization becomes indispensable for many wireless sensor network applications. However, due to the resource constraints on sensor nodes, traditional time synchronization protocols (e.g., NTP [65]) cannot be directly applied in sensor networks. Recently, several time synchronization protocols (e.g., [26, 31, 63, 55, 67, 87, 72, 42]) have been proposed for wireless sensor networks in benign environments. However, without addressing security, all the time synchronization techniques in wireless sensor networks cannot survive the malicious attacks in hostile environments. As many other techniques, security is not the top priority when designing time synchronization protocols for wireless sensor networks. This dissertation focuses on developing secure and resilient time synchronization techniques for wireless sensor networks to work in hostile environments.

1.1 Motivation

In hostile environments, an adversary may certainly attack the time synchronization protocol due to its importance. Note that all time synchronization protocols rely on *time-sensitive* message exchanges. To mislead these protocols, the adversary may forge or modify time synchronization messages, jam the communication channel to launch Denial of Service (DoS) attacks, and launch pulse-delay attacks [29] by first jamming the receipt of time synchronization messages and then later replaying buffered copies of these messages. The adversary may also launch wormhole attacks [43] by creating low latency and high bandwidth communication channels between different locations in the network, and (selectively) delay or drop time synchronization messages transmitted through the wormholes. The adversary may use Sybil attacks [24, 69], where one node presents multiple identities, to defeat typical fault tolerant mechanisms. Though message authentication can be used to validate message sources and contents, it cannot validate the *timeliness* of messages, and thus is unable to defend against all of these attacks.

Moreover, the adversary may compromise some nodes, and exploit the compromised nodes in arbitrary ways to attack time synchronization. For example, the adversary may instruct the compromised nodes to (selectively) delay or drop time synchronization messages, and launch Sybil attacks [69] using the identities and keying materials of compromised nodes if message authentication is enabled. The compromised nodes may collude with each other to cause the worst damage to the network.

In this dissertation, we propose a suite of secure time synchronization techniques to achieve secure time synchronization between two neighbor nodes, among a group of sensor nodes, and in a whole sensor network, respectively.

1.1.1 Secure Single-hop Pair-wise Time Synchronization

Single-hop pair-wise time synchronization aims at obtaining a high-precision time synchronization between pairs of sensor nodes. Researchers have proposed two approaches to achieve single-hop pair-wise time synchronization: *receiver-receiver synchronization* (e.g., RBS [26]), in which a reference node broadcasts a reference packet to help pairs of receivers to identify the clock differences, or *sender-receiver synchronization* (e.g., TPSN [31], FTSP [63]), where a sender communicates with a receiver to estimate the clock difference.

In receiver-receiver synchronization (e.g., RBS [26]), a compromised reference node may provide different non-compromised nodes different time values about the receipt of the reference packet. Moreover, an adversary can compromise it by simply launching a pulse-delay attack [29] or wormhole attack [44] against one of the nodes to manipulate the packet transmission delay, so that the two nodes receive the reference packet at different times. In FTSP [63], a sender-receiver synchronization, one node passes its own time to the other by directly sending a MAC layer timestamped packet to the latter. This works well in benign environments, as demonstrated in [63]. However, in hostile environments, it suffers from the same problems mentioned above. TPSN uses a sender-receiver approach (through one request and one reply message) to help the sender obtain its clock difference from the receiver [31]. A malicious node may distort the pair-wise time synchronization by jamming the signal between two normal nodes and then replaying the delayed signal to introduce synchronization error [29]. Even an external attacker can launch this attack. TPSN was later improved with security in Secure Pair-Wise Synchronization (SPS) [29] to deal with pulse-delay and wormhole attacks. Specifically, it authenticates the messages being exchanges, and uses the timestamp information to estimate both the clock difference and the message transmission delay. Pulse-delay and wormhole attacks that manipulate packet transmission delay will introduce extra delay in message transmission, and will be detected.

Medium Access Control (MAC) layer timestamping has been widely accepted as an effective way to reduce the synchronization error during the message exchanges since it was proposed in [31]. To ensure the integrity of pair-wise time synchronization, we may authenticate a synchronization message by adding a Message Integrity Code (MIC) once the MAC layer timestamp is added. This, however, introduces a potential problem due to the extra delay required by the MIC generation: It is necessary to have a MAC layer timestamp that marks the exact transmission time of a certain bit in the message at the sender's side, but the MIC generation and insertion require extra delay and have to be done after the timestamp is inserted into the message.

The delay introduced by MIC generation using software (e.g., TinySec [49]) can be tolerated for sensor platforms with low data rate radio components, such as MICA2 motes (38.4 kbps data rate). However, with the increased data rate on recent sensor platforms with IEEE 802.15.4 compliant radio components (250 kbps data rate [46]), such as MICAz and TelosB motes, there is not enough time to generate and insert the MIC before the transmission of the MIC bytes due to the delay introduced by the MIC calculation [29].

In Chapter 3, we develop a secure single-hop pair-wise time synchronization technique by adopting the SPS approach [29]. Due to the long delay on generating the message integrity code by using software, SPS approach [29] can only work on low data rate motes such as MICA2, while it does not support high data rate motes such as MICAz and TelosB. Unlike the SPS approach, our approach uses a *hardware-assisted, authenticated MAC layer timestamping* technique to handle high data rate such as those produced by MICAz and TelosB motes (in contrast to those by MICA2 motes).

1.1.2 Fault-tolerant Cluster-wise Time Synchronization

In wireless sensor networks, it is usually necessary to have a cluster of nodes share a common view of a local clock time, so that the nodes can coordinate their actions. For example, in time slotted MAC protocols, the multiple access to the shared communication medium is achieved by assigning time slots to a group of nodes. Therefore, the sensor nodes require a synchronized clock to access their time slots without colliding with other nodes. As another example, to increase the energy efficiency, a cluster of sensor nodes may frequently switch into power-saving sleep mode at the same time [100]. They also require a common clock to coordinate their sleep/listen periods. In benign environments, such a local common clock can be easily achieved by having all the nodes synchronize to a given node. However, in hostile environments where some nodes may be compromised, it is quite challenging to synchronize the clocks among a cluster of nodes. Indeed, none of the aforementioned time synchronization protocols can survive malicious actions by compromised nodes. A compromised node may disrupt the time synchronization by sending different time values to non-compromised nodes. For example, when RBS [26] is used for pair-wise time synchronization, a compromised node may provide different non-compromised nodes different time values about the receipt of the reference packet.

To provide secure and resilient cluster-wise time synchronization, it is natural to consider fault-tolerant time synchronization techniques, which have been studied extensively in the context of distributed systems (e.g. [78, 53, 39, 22, 58, 59, 89, 84, 85, 71, 13, 50, 86, 96]). However, traditional fault-tolerant time synchronization techniques are not directly applicable to wireless sensor networks. These techniques were developed for distributed systems that do not have the same resource constraints as wireless sensor networks. All of these techniques involve heavy communication among the nodes, and sometimes heavy computation as well. This is because these techniques either use digital signatures (e.g., HSSD [22], CSM [53]) or multiple copies of messages (e.g., COM, CNV [53]) to prevent a malicious node from modifying or destroying clock information sent by nonfaulty nodes without being detected. Digital signature is usually not practical in resource constrained wireless sensor networks. Even when digital signature is used, for example, in HSSD [22], each node still needs to send a message to every other node in each synchronization round, resulting in at least $O(n^2)$ communication complexity, where n is the number of nodes. Some schemes (e.g., HSSD [22], ST [89]) require that all nodes that receive certain messages process and forward these messages to all the other nodes immediately, resulting in a high probability of message collisions if used in wireless sensor networks.

In Chapter 4, we present a novel fault-tolerant cluster-wise time synchronization scheme for clusters of nodes in wireless sensor networks, where the nodes in each cluster can communicate through broadcast. The proposed scheme guarantees an upper bound of clock difference between any nonfaulty nodes in a cluster, provided that the malicious nodes are no more than one third of the cluster. Unlike the traditional fault-tolerant time synchronization approaches, the proposed technique does not introduce collisions between synchronization messages, nor does it require costly digital signatures.

1.1.3 Secure and Resilient Global Time Synchronization

A number of time synchronization protocols (e.g., [26, 31, 63, 55, 79, 67, 87, 72, 42, 21, 35, 90]) have been proposed for wireless sensor networks to achieve *global* time synchronization. Most of the global time synchronization protocols (e.g., [26, 31, 87]) establish multi-hop paths in a wireless sensor network, so that the other nodes can synchronize their clocks to the source based on these paths and the single-hop pair-wise clock differences between adjacent nodes in these paths. Alternatively, diffusion based global synchronization protocols [55] achieve global synchronization by spreading local synchronization information to the entire network.

All these techniques assume benign environments. Though it is possible to use authentication to defend against external attacks, an attacker may still attack time synchronization through compromised nodes. When a pair of nodes are synchronized through a multi-hop path (e.g., [26, 31, 87]), a compromised node in the path can introduce an arbitrary error. This implies global time synchronization using multi-hop paths is vulnerable to compromised nodes. When the diffusion based global time synchronization techniques [55] are used, compromised nodes may fluctuate their clock information periodically to prevent the convergence of the clocks. Therefore, a secure and resilient global time synchronization is necessary for wireless sensor networks to work in hostile environments.

In Chapter 5, we propose two secure and resilient time synchronization schemes, *level-based time synchronization* and *diffusion-based time synchronization*. The basic idea of both schemes is to provide redundant ways for one node to synchronize its clock with another far-away node, so that it can tolerate partially missing or false synchronization information provided by compromised nodes. The level-based scheme builds a level hierarchy in the sensor network, and then synchronizes the whole network level by level. The diffusion-based scheme allows each node to diffuse its clock to its neighbor nodes after it has been synchronized. Both schemes are secure against external attacks and resilient against compromised nodes.

To achieve global time synchronization, we first choose to propagate the global synchronization information using authenticated unicast messages, provided each pair of nodes share a secure key. However, through the simulation results, we found out that in each round of global time synchronization, the communication overhead as well as the message collisions is quite huge. Thus, it can hardly be extended for large sensor networks. To solve this problem, in Chapter 6, we develop a secure and resilient global time synchronization using broadcast authentication based on a novel use of the μTESLA [76] broadcast authentication protocol for *local authenticated broadcast*. In each round of global time synchronization, each node only broadcasts one message. Thus, both the message overhead and the message collisions are reduced dramatically.

1.2 Summary of Contributions

In this dissertation, we present three techniques to provide secure and resilient time synchronization to work in different scopes of wireless sensor networks. The summary of these techniques are as follows:

- *Secure Single-hop Pair-wise Time Synchronization:* We develop a secure single-hop pair-wise time synchronization technique by adopting the SPS approach [29]. SPS uses a random nonce to prevent replay of a previously transmitted reply message. In our case, we simply use the sender's timestamp in the reply message to prevent replay attack, so that we can further reduce the message size. SPS approach uses software security (e.g., TinySec [49]) to authenticate the MAC layer timestamping in synchronization messages. The message integrity code using software can be available before the radio component sends the corresponding bytes in the message for sensor platforms with low data rate. However, with the increased data rate on recent sensor platforms, such as MICAz and TelosB motes, there is not enough time to generate and insert the MIC before the transmission of the MIC bytes due to the delay introduced by the MIC calculation [29].

Unlike SPS approach, we propose a *hardware-assisted, authenticated MAC layer timestamping* technique to handle high data rate such as those produced by MICAz and TelosB motes (in contrast to those by MICA2 motes) with the hardware security support in their radio components. We implement the proposed technique on MICAz motes [20] running TinyOS [41]. The secure single-hop pair-wise time synchronization will serve as the building block to achieve the secure and resilient global time synchronization.

- *Fault-tolerant Cluster-Wise Time Synchronization:* This technique provides a novel fault-tolerant cluster-wise clock synchronization for a cluster of sensor nodes, where the nodes in each cluster can communicate with each other directly through broadcast. In each round of time synchronization, only one node serves as the *synchronizer*, and only one authenticated synchronization message is broadcast. Thus, our scheme can avoid the message collision problem. The proposed scheme exploits a recently proposed local broadcast authentication technique for sensor networks, which is purely based on symmetric cryptography [104], thus avoiding the costly digital signature for message authentication. Our analysis shows that the proposed scheme guarantees an upper bound on the clock difference between nonfaulty nodes when no more than $1/3$ of the nodes are compromised and collude with each other.

- *Secure and Resilient Global Time Synchronization:* This research resulted in two secure and resilient time synchronization schemes: level-based and diffusion-based time synchronization. The level-based scheme builds a level hierarchy in the wireless sensor network, and then synchronizes the whole network level by level. The diffusion-based scheme allows each node to diffuse its clock to its neighbor nodes after it has synchronized to the source node. Our basic idea is to provide redundant ways for each node to synchronize its clock with the common source, so that it can tolerate partially missing or false synchronization information provided by the malicious nodes. To improve the performance and the resilience of our techniques, we propose to deploy multiple source nodes in the network.

  We first propose to use authenticated unicast messages to distribute the synchronization information by assuming that each two neighbor nodes can share a secret pair-wise key. However, because each node needs to send one message to each neighbor node in each round of time synchronization, the communication overhead is quite high, and it may cause potential huge message collisions, especially when neighboring nodes intend to send several messages at the same time. Therefore, the technique using unicast authentication can hardly be extended for large sensor networks.

To solve this problem, we later develop a secure and resilient global time synchronization protocol, *TinySeRSync*, using broadcast authentication based on a novel use of the μTESLA broadcast authentication protocol for *local authenticated broadcast*. We resolve the conflict between the goal of achieving time synchronization with μTESLA-based broadcast authentication and the fact that μTESLA requires loose time synchronization. We implement the proposed TinySeRSync protocol on MICAz motes running TinyOS and perform a thorough evaluation through field experiments in a network of 60 MICAz motes. The evaluation results indicate that TinySeRSync is a practical system for secure and resilient global time synchronization in wireless sensor networks.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 presents a general clock model and discusses the related work on time synchronization. Chapter 3 presents the secure single-hop pair-wise time synchronization scheme. Chapter 4 gives out our fault-tolerant cluster-wise time synchronization technique. We present the secure and resilient global time synchronization schemes using unicast authentication in Chapter 5, and the schemes using broadcast authentication in Chapter 6. Chapter 7 concludes this dissertation and points out some future research directions.

Chapter 2

Background

In this chapter, we first introduce a typical clock model used in distributed systems. Then, we present several traditional time synchronization techniques. Next, we discuss the time synchronization schemes proposed for wireless sensor networks. Finally, we give out a number of performance metrics that can be used to evaluate and compare secure time synchronization techniques in wireless sensor networks.

2.1 Clock Model

We introduce a typical clock model used in distributed systems, which is adapted from [22]. The wireless sensor networks usually use the same clock model. A clock is an instrument for measuring time. We first make a distinction between *real time* and *clock time*. Real time is an assumed Newtonian time frame that may not be directly observable, and clock time is the time that can be observed on the clocks. We use lowercase letters to denote the variables and constants about real time, and uppercase letters to denote those about clock time.

A *local clock* C can be considered a mapping from real time to clock time, i.e., $T = C(t)$ is the clock time at the real time t. When we speak of “a clock drifting from real time”, we mean that the difference between a clock and real time may gradually increase. A clock C is considered *well-behaved* if its rate of drift from the real time is bounded by a constant $\rho > 0$ for all the real time points t_1 and t_2, where $t_1 < t_2$:

$$\frac{t_2 - t_1}{1 + \rho} < C(t_2) - C(t_1) < (1 + \rho)(t_2 - t_1). \tag{2.1}$$

The rate of drift between any two well-behaved clocks is bounded by $\lambda = \rho(2 + \rho)/(1 + \rho)$, which is less than 2ρ. The drift rate of a clock may fluctuate over time due to aging, changes in the environment, and by other factors external to the oscillator.

When it is difficult to directly discipline a local clock, a node may construct a software clock. A software clock can be considered a mapping from a local clock time to a software clock time, which can provide a synchronized clock time by adjusting the parameters of the software clock.

In all the synchronization schemes that depend on exchanging messages, due to the delay uncertainty, the synchronization error consists of the following basic components [26]:

- send time: It is the time spent to construct a message at the sender. It includes the operating system overhead (e.g., context switches) and the time to send the message to network interface.
- access time: It is the time delay for accessing the physical channel. It is specific to the MAC layer protocol in use. Contention-based MAC layer protocols must wait for the channel to be clear before transmitting, and retransmit in case of collision. TDMA-based MAC layer protocols require the sender to wait for its slot before transmitting.
- propagation time: It is the time for a message to propagate from the sender to the receiver.
- receive time: It is the time for the receiver to receive and transfer the message for the host.

Existing time synchronization techniques vary primarily in their methods for estimating and eliminating the above sources of error.

Another thing that affect the precision of a synchronization algorithm is the clock drift between two synchronization points. Due to the clock drift, a clock may drift from the real time before the next round of time synchronization. To decrease the maximum clock difference from the real time, a clock may run time synchronization process more frequently, or estimate and correct the clock drift if the clock has a long-term stable drift rate.

2.2 Traditional Time Synchronization

Time synchronization problem has been investigated thoroughly in Internet and distributed computer systems. Next, we discuss some representative time synchronization mechanisms.

Global Positioning System (GPS) GPS [3, 98] is a constellation of satellites operated by the U.S. Department of Defense. GPS is originally intended to be used for precise positioning through the determination of pseudo-ranges from the satellites to the ground based receiver. The key idea is that by measuring the time of flight of a radio signal from 4 or more satellites to the receiver, the position of the receiver may be accurately determined. In addition, the time difference of the receiver from the GPS clock time may be calculated. Therefore, the velocity of the receiver and the time frequency offset of the receiver may be ascertained. GPS can provide a time accuracy of several nanoseconds.

NTP In Internet, computers can obtain a synchronized Internet time by using Network Time Protocol (NTP) [65] protocol. NTP organizes all the computers in a client/server structure. Primary servers synchronize to national reference clock sources via radio, satellite and modem. Then, secondary servers and clients synchronize to primary servers via hierarchical subnet. The reliability is assured by redundant servers and diverse network paths. Several engineered algorithms have been proposed to reduce jitter, select from multiple sources and avoid improperly operating servers. The system clock of a computer is disciplined in time and frequency using an adaptive algorithm responsive to network time jitter and clock oscillator frequency wander. NTP provides accuracies of low tens of milliseconds on WANs, sub-milliseconds on LANs, and sub-microseconds using a precision time source such as a cesium oscillator or GPS receiver.

Probabilistic-based Synchronization Cristian [19] proposed a probabilistic method to read remote clocks in distributed systems that are subject to unbounded random communication delays. When a process wants to synchronize to a remote process, it sends a time request to the remote process, and calculates the request's round-trip time as the difference between the time when it initiates the request and the time when it receives the reply from the remote process. The reply contains the time when the remote process sends the reply. Then, the process adjusts its clock time to the sum of the time contained in the reply and half the round-trip time. Due to the non-deterministic message delay, to reduce the synchronization error, a process needs to perform multiple such trails and chooses the trial with the mininum round-trip time to synchronize its clock. Cristian's method is probabilistic because it does not guarantee a processor can always synchronize to a remote process with an a priori specified precision. Therefore, to increase the probability of success to achieve a given precision, a process needs to increase the number of trials on estimating the remote process's clock time.

TEMPO Gusella and Zatti [38] proposed a centralized time synchronization service for the Ethernet local area network. A master node first measures the time differences between its local clock and those of other slave nodes. The master node computes the network time as the average of the times provided by normal clocks, and then sends to each slave node the correction that should be performed on its clock. This process is repeated periodically. It assumes the master node is always trusted. The similar idea can be used in sensor networks to achieve time synchronization in a group of nodes that can directly communicate with a sink node or a normal sensor node.

Fault-tolerant Time Synchronization In distributed systems, fault-tolerant time synchronization has undergone substantial research (e.g. [52, 78, 53, 39, 22, 58, 59, 89, 84, 85, 71, 13, 50, 86, 96]). These techniques take either a *software* or a *hardware* approach [78]. Hardware-based techniques require a synchronization circuitry continuously monitor all the clocks [78], and thus cannot be used in sensor networks. Software-based techniques can be further classified into convergence-averaging (e.g., CNV [53], LL [58, 59]), convergence-non-averaging (e.g., HSSD [22], ST [89]), or consistency algorithms (e.g., COM, CSM [53]). Some software-based (or hardware-assisted, hybrid) techniques [77] use hardware to generate timestamps, and thus can reduce the uncertainty involved in time synchronization. A common theme of these techniques is to use redundant messages to deal with malicious participants that may behave arbitrarily.

All the traditional time synchronization techniques cannot be directly applied into sensor networks, mainly due to the resource constraints of sensor nodes. For example, the GPS receiver is too large, expensive, and power-hungry for small, cheap, and power constrained sensor nodes. Moreover, GPS requires a clear sky view, which is not always available in some areas, such as inside of buildings or underwater. Because NTP does not consider the energy and computation limitations of sensor nodes [27], it is infeasible to implement NTP in sensor networks. NTP uses several engineered algorithms (i.e., data-filtering algorithm, peer-selection algorithm and combining algorithm) to reduce jitter, increase the robustness and avoid improperly operating servers. Such algorithms are computationally intensive and assumes the CPU is always available to frequently discipline the oscillator. However, the CPU cycles in sensor nodes are also a scarce resource, and the sensor nodes cannot spend all the CPU cycles on the time synchronization. Consider Cristian's probabilistic method. It requires a large number of message exchanges, which introduces a high communication overhead for resource constrained sensor nodes. Moreover, it can only provide a probabilistic time synchronization with a given precision. When TEMPO is deployed to achieve a network-wide time synchronization, the bounds on the synchronization accuracy will increase a lot due to the delay uncertainty in multi-hop message communication.

The traditional fault-tolerant time synchronization schemes in wired networks usually assume there is unlimited computing resource and network bandwidth, and thus are not suitable for sensor networks. These schemes usually have very high communication overhead (especially the consistency-based approaches such as COM and CSM). Moreover, to prevent malicious participants from forging messages, these schemes use either digital signatures (e.g., CSM [53], HSSD [22]), or a broadcast primitive that requires simultaneous broadcast from multiple nodes, which will result in message collisions in wireless sensor networks.

2.3 Time Synchronization in Wireless Sensor Networks

Recently, several time synchronization protocols (e.g., [26, 27, 21, 31, 63, 72, 87, 55, 90, 67, 35, 42, 79, 64]) have been proposed for sensor networks to achieve *pair-wise* and/or *global* time synchronization. Pair-wise time synchronization aims to obtain a high-precision time synchronization between pairs of sensor nodes, while global time synchronization aims to provide network-wide time synchronization in a sensor network.

Reference Broadcasting Elson et al. developed the Reference Broadcast Synchronization (RBS) scheme for pair-wise as well as multi-domain time synchronization [26], which eliminates the uncertainty of send time and access time from the clock reading error by using a reference broadcast node. In RBS, one sender broadcasts a single pulse, two receivers can calculate their relative clock difference by exchanging the receiving time of the pulse from the sender. In addition, the sender can broadcast a number of pulses to improve the precision between the receivers. They also propose to use the least square linear regression technique to estimate the clock frequency difference. RBS can provide an average synchronization error of 11 μs by using 30 broadcasts. Based on RBS, Palchaudhuri et al. [72] proposed a probabilistic time synchronization which can reduce the communication overhead; however, it can only probabilistically guarantee the required time precision.

Timing-sync Protocol Generiwal et al. proposed a hierarchical time synchronization scheme named TPSN for sensor networks [31], assuming time synchronization messages are timestamped at MAC layer. In the pair-wise time synchronization, one sender synchronizes itself to a receiver by exchanging one pair of messages. If the two exchange messages can be timestamped at MAC layer right before being sent out, TPSN can provide a higher time precision than RBS. TPSN can provide a global time synchronization in two phases: level discovery and synchronization. The goal of level discovery phase is to build a spanning tree topology in the network, where each node is assigned a level. The tree topology is rooted at one source node, which is assigned level 0. In the synchronization phase, a level i node synchronizes to a neighbor node at level $i-1$. In the end, all the nodes are synchronized to the source node, and the global time synchronization is achieved. Generiwal et al. [30] also proposed a synchronization scheme to estimate the long-term clock changes and to reduce the energy consumption in duty-cycling MAC layer in sensor networks.

Flooding Time Synchronization Mar$\acute{o}$ti et al. [63] proposed a flooding time-synchronization protocol to synchronize a whole network. The node with the lowest node ID is elected as the leader that serves as the reference node. The leader periodically floods the network with a synchronization message that contains the leader's current time. Nodes that have not received this message record the time stamp in the message and the receiving time of the message, and broadcast the message to their neighbors. It uses MAC layer time stamping to minimize the delay uncertainty. Each node collects eight messages and uses the linear regression to estimate the offset and the frequency difference to the leader.

Global Time Synchronization Li and Rus proposed a global time synchronization technique based on local diffusion of clock information [55]. Nodes achieve global synchronization by flooding their neighbors with information about its local clock value. After each node have received the clock values of all its neighbors, the node can use a derived consensus value to adjust its clock. They presented two protocols for both synchronous and asynchronous situations. In the synchronous protocol, all the nodes can execute each round of synchronization at the same time. Each node adjusts its clock value by a factor proportional to the clock difference from each neighbor node. The factor indicates the weight of a neighbor on the node's clock adjustment. In the asynchronous protocol, a node starts one round of synchronization by requesting local clock values from all its neighbors. Then, it updates its clock to the average of its clock value and the clock values from all the neighbors. All the nodes may start the time synchronization process at different times. Both protocols can converges to the average value of the clock readings in the network, within a certain error range. By using their protocols, dense networks converge faster, and with lower variation in convergence time, than sparse networks.

Time Diffusion Synchronization Su and Akyildiz proposed a time diffusion synchronization (TDP) protocol to support a network-wide time synchronization [90]. Initially, a set of master nodes are elected based on its remaining power energy. Then, each master node establishes a tree hierarchy. After obtaining the round-trip time to each neighbor node, a master node calculates and broadcasts the average and standard deviation of the message delay to all neighbors. A neighbor node becomes a diffused node based on its remaining power energy and its clock property, and repeat the procedure as the master nodes. The average delay and standard deviation are summed up along the path from the master nodes. The diffusion procedure stops at a given number of hops from the master nodes. One node may appear in multiple master nodes's trees, so it adjusts its clock according to the clock differences from all these master nodes. To achieve load balance, all the master nodes are reelected in a given interval. The diffused nodes are reelected in a smaller time interval. This protocol contains many algorithms on electing master nodes and synchronizing sensor nodes; however, these complex algorithms may be too heavy for resource-constrained sensor nodes.

Tiny-sync and Mini-sync Sichitiu et al. [87] developed two lightweight pair-wise synchronization schemes, Tiny-Sync and Mini-Sync, to deterministically estimate the bounds on both the relative clock drift and offset between two sensor nodes. Both schemes use multiple round-trip measurements and a line-fitting technique to obtain the clock offset and the relative clock drift of two nodes. Tiny-Sync uses a heuristic to keep only two measurements in storage, but only achieves a suboptimal solution. Mini-Sync can provide an optimal solution with increased computation and storage overhead.

Lightweight Tree-Based Synchronization Greunen and Rabaey claimed that the maximum time accuracy required in sensor networks is relatively low (within a fraction of one second), and proposed two lightweight global time synchronization schemes in sensor networks [35]. Both schemes synchronize all the sensor nodes to some reference node(s) in the sensor network. The first scheme is centralized and needs to construct a spanning tree rooted at the reference node. The reference node is responsible for initializing the synchronization process with a time interval which is decided by the depth of the tree and the required precision. The second scheme performs in a distributed fashion. When a node needs to synchronize its clock, it sends a synchronization request to the reference node by using any available routing protocol. Then all the nodes along the path from the reference node to the requesting node must be synchronized before the requesting node. To reduce the synchronization overhead, the authors proposed to aggregate several requests along the same path into one request.

TSync Dai and Han [21] proposed two time synchronization schemes: the Hierarchy Reference Time Synchronization protocol (HRTS) for proactive synchronization of the whole network, and the Individual-based Time Request protocol (ITR) for reactive synchronization of individual nodes. Both protocols need an independent radio channel for synchronization messages to avoid the inaccuracies due to the variable delay introduced by message collisions. In HRTS, a spanning tree rooted at a reference node is constructed. Then, the reference node use the reference broadcasting techniques [26] to synchronize the network. The ITR protocol differs from the HRTS protocol in that synchronization is initiated by any node as opposed to a designated reference node.

Time Synchronization for Ad-hoc Networks Römer proposed a time synchronization scheme for wireless ad-hoc networks [79]. The basic idea is not to synchronize the local clocks of nodes, but instead generate time stamps using unsynchronized local clocks. When a message containing a time stamp is transmitted between two nodes, the time stamp is first transformed from the sender's local time to the standard Coordinated Universal Time (UTC), and then to the receiver's local time. The final timestamp is expressed as an interval with a lower bound and an upper bound. Meier et al. [64] improved Römer's protocol by providing a tight bound on the transformed time interval on the receiving node.

Time Synchronization in IEEE 802.11 The IEEE 802.11 standard [9] requires time synchronization in wireless networks for keeping hopping synchronized and other functions like power saving. In an infrastructure BSS (Basic Service Set), a simple master/slave protocol is used to synchronize the stations to an access point (AP). The AP transmits periodic beacon frames to the stations, which adjust their clocks to the clock value in the frames from the AP. In an independent BSS (IBSS), time is divided into beacon intervals. At the beginning of each interval, each station calculates a random delay and is scheduled to transmit a beacon when the delay timer expires. If a beacon arrives before the random delay timer has expired, the station cancels the pending beacon transmission and the remaining random delay. Upon receiving a beacon, a station sets its clock to the timestamp of the beacon if the value of the timestamp is later than the stations clock time. It guarantees that clocks only move forward and never backward. However, this approach has scalability problem that the fastest station will be out of synchronization when the number of stations increases. ATSP was proposed in [45] to solve the scalability problem. The basic idea is to let the fastest station to compete for beacon transmission every beacon interval and let other stations to compete only occasionally.

All of the above techniques assume benign environments, but cannot survive malicious attacks from compromised nodes. There have been several recent studies for secure time synchronization in sensor networks [29, 93, 62, 88]. Manzo et al. discussed a few attacks against existing time synchronization protocols and several countermeasures to protect time synchronization [62]. However, there was no mechanism to authenticate the timeliness of synchronization messages, and thus no protection against, for example, pulse-delay attacks [29] and worm-hole attacks [44], in which the adversary may delay authenticated synchronization messages.

Song et al. investigated countermeasures against attacks that mislead sensor network time synchronization by delaying synchronization messages [88]. They proposed two methods for detecting and tolerating delay attacks: one transforms attack detection into statistical outliers detection, and the other detects attacks by deriving the bound of the time difference between two nodes through message exchanges. Unfortunately, [88] only addresses synchronization of neighbor nodes, but does not support global time synchronization in multi-hop sensor networks.

Ganeriwal et al. proposed a secure single-hop pair-wise synchronization (SPS) technique [29], which provides authentication for medium access control (MAC) layer timestamping by adding timestamp and message integrity code (MIC) as the messages being transmitted. This approach works for low data rate sensor radios (e.g., CC1000 on MICA2 motes with 38.4Kbps data rate); however, it cannot keep up with recent IEEE 802.15.4 [46] compliant sensor radios such as CC2420 on MICAz and TelosB, whose data rate is 250Kbps.

Though it is possible to use authentication to defend against external attacks, an attacker may still attack time synchronization through compromised nodes. These compromised nodes may drop, modify, forge, or replay the synchronization messages. We don't consider the physical layer attacks, such as signal jamming attacks.

2.4 Evaluation Metrics for Secure Time Synchronization

Given a secure time synchronization protocol, which targets at preventing or mitigating the attacks in time synchronization, we can use the following metrics to evaluate its performance in both benign and hostile environments.

- *synchronization precision:* It is the maximum clock difference between any two sensor nodes in a whole network. It is a metric closely related to the *synchronization error*, which is about the clock offset of a single node [81]. In hostile environments, we only care about the synchronization precision between normal nodes.

- *synchronization rate:* It is the percentage of sensor nodes in a sensor network that can correctly obtain a synchronized clock time.

- *memory overhead:* It is the size of memory allocated for storing the messages related to time synchronization in each sensor node. Currently, memory is still critical for resource constrained sensor nodes.

- *convergence time:* It has different meanings for external synchronization and internal synchronization.
  - *external synchronization* When all the clocks in the network are synchronized to an external clock source, the convergence time is the time interval between the start point of the synchronization process and the time point when the last sensor node that can be synchronized synchronizes its clock.
  - *internal synchronization* When all the nodes in a network need to agree on a consistent clock time without the help from an external clock source, the convergence time is the time interval between the start point of the synchronization process and the time point when the predetermined synchronization precision is achieved.
- *communication overhead:* It is the number of messages sent for time synchronization. The synchronization information may be piggy-backed in the messages for other applications, or sent by dedicated synchronization messages. The "piggy-back" method can avoid additional messages for synchronization, but it may not provide a synchronized clock on demand, since it depends on the other applications. The communication overhead is related to the synchronization precision achieved.

In the following chapters, we will use the above metrics to study the performance of the proposed secure time synchronization techniques.

Chapter 3

Secure Single-Hop Pair-Wise Time Synchronization

In wireless sensor networks, existing pair-wise or global time synchronization techniques are all based on *single-hop* pair-wise time synchronization, which discovers the clock difference between two neighbor nodes that can communicate with each other directly. In single-hop pair-wise time synchronization, Medium Access Control (MAC) layer timestamping has been widely accepted as an effective way to reduce the synchronization error during the message exchanges since it was proposed in [31].

To ensure the integrity of pair-wise time synchronization, we may authenticate a synchronization message by adding a Message Integrity Code (MIC) once the MAC layer timestamp is added. This, however, introduces a potential problem due to the extra delay required by the MIC generation: It is necessary to have a MAC layer timestamp to mark the exact transmission time of a certain bit in the message at the sender's side, but the MIC generation and insertion require extra delay and have to be done after the timestamp is inserted into the message.

The delay introduced by MIC generation using software (e.g., TinySec [49]) can be tolerated for sensor platforms with low data rate radio components, such as MICA2 motes (38.4 kbps data rate). However, with the increased data rate on recent sensor platforms with IEEE 802.15.4 compliant radio components (250 kbps data rate [46]), such as MICAz and TelosB motes, there is not enough time to generate and insert the MIC before the transmission of the MIC bytes due to the delay introduced by the MIC calculation [29].

In this chapter, we present a secure single-hop pair-wise time synchronization technique [94] by adopting a Secure Pair-Wise Synchronization (SPS) [29] technique, which provides a software based, MAC layer timestamp authentication for low data rate radio components. We propose a *hardware-assisted, authenticated MAC layer timestamping* technique to handle high data rate in a specific target of the IEEE 802.15.4 compliant radio component ChipCon CC2420 [6], which is commonly used in recent sensor platforms such as MICAz and TelosB motes. CC2420 features hardware security support for data encryption and data authentication. We implement the proposed technique on MICAz motes running TinyOS [41].

3.1 Attacks in Single-Hop Pair-Wise Time Synchronization

Most of the single-hop pair-wise time synchronization schemes suffer the following attacks in hostile environments.

- *pulse-delay attack:* A malicious node may distort the pair-wise time synchronization by jamming the signal between two normal nodes and then replaying the delayed signal to introduce synchronization error [29]. Even an external attacker can launch this attack.

- *malicious reference:* This attack is specific to receiver-receiver synchronization (e.g., RBS [26]). A compromised reference node may provide different non-compromised nodes different time values about the receipt of the reference packet.

- *replay attack:* A malicious node may launch replay attacks by recording the current synchronization messages from other nodes and impersonating these nodes to send the buffered messages later. This type of attack can usually be defeated through the use of freshness token such as a sequence number.

- *Sybil attack:* A malicious node may attempt to forge multiple identities by launching Sybil attacks [69]. If colluding malicious nodes can exchange their keying materials, one malicious node may impersonate other remote malicious nodes in its local network [73].

- *wormhole attack:* In sensor networks, remote malicious nodes may pretend to be in normal nodes' local area through wormholes [44]. Thus, time synchronization messages at a remote area may be tunneled to local area to interrupt the local time synchronization process.

An attacker can also launch signal jamming attacks, which can block normal nodes for receiving any synchronization messages. Because no scheme that requires inter-node communication can survive such attacks, we did not consider such attacks in our model.

In the following, we describe our secure single-hop pair-wise time synchronization technique that can achieve correct time difference between two normal neighbor nodes.

3.2 Secure Single-Hop Pair-Wise Time Synchronization

The goal of secure single-hop pair-wise time synchronization is to ensure two neighbor nodes can obtain their clock difference through message exchanges in a secure way. This requires the authentication of the source, the content (i.e., the timing information), and the timeliness of each message used for such synchronization.

In the following, we first discuss how we provide authentication of the source and the timing information in synchronization messages, and then describe a secure two-way pair-wise time synchronization protocol for a node to obtain the clock difference from a neighbor node.

3.2.1 Authenticated MAC Layer Timestamping

MAC layer timestamping has been widely accepted as an effective way to reduce the synchronization error during the message exchanges since it was proposed in [31]. By adding (on the sender's side) and retrieving (on the receiver's side) timestamps in the MAC layer, this approach avoids the uncertain delays introduced by application programs and medium access, and thus has more accurate synchronization precision.

To ensure the integrity of pair-wise time synchronization, we may authenticate a synchronization message by adding a MIC once the MAC layer timestamp is added, assuming the two nodes performing pair-wise synchronization share a secret pair-wise key through, for example, TinyKeyMan [57]. This, however, introduces a potential problem due to the extra delay required by the MIC generation: It is necessary to have a MAC layer timestamp that marks the exact transmission time of a certain bit in the message at the sender's side, but the MIC generation and insertion require extra delay and have to be done after the timestamp is inserted into the message.

The delay introduced by MIC generation and insertion can be tolerated for sensor platforms with low data rate radio components, such as MICA2 motes. In an earlier study [29], Ganeriwal et al. attempted to provide authenticated MAC layer timestamping for MICA2 motes (38.4 kbps data rate) by generating MIC on the fly. Specifically, when the radio component of a sensor node begins to transmit the first byte of a synchronization message, it appends the current timestamp into the message, calculates the MIC, and appends the MIC into the message being transmitted. Due to the low data rate (38.4 kbps), the MAC layer timestamp and the MIC can be added into the packet before the corresponding bytes are transmitted [29]. However, with the increased data rate on recent sensor platforms with IEEE 802.15.4 compliant radio components (250 kbps data rate [46]), there is not enough time to generate and insert the MIC before the transmission of the MIC bytes due to the delay introduced by the MIC calculation [29].

We propose a prediction-based approach to address the above problem. In the following, we describe our approach, with a specific target of the IEEE 802.15.4 compliant radio component ChipCon CC2420 [6], which is commonly used in recent sensor platforms such as MICAz and TelosB motes. We also assume the sensor nodes use TinyOS [41], the open source operating system for networked sensor nodes.

Prediction-Based MAC Layer Timestamping and Hardware-Assisted Authentication

We observe that the code for generating a MIC is deterministic, and the time required for a MIC generation for messages with a given length (or, more precisely, a given number of blocks) is fixed. In addition, the process to transmit a packet (starting from observing the channel vacancy to the actual transmission of data payload) in CC2420 is also deterministic. Thus, when we put a timestamp into a synchronization message to be authenticated in the MAC layer, we may predict the time required by MIC generation and at the same time predict the delay between the start of transmission and the transmission a given bit in the packet.

Let us review how a sensor node (such as a MICAz mote) equipped with a CC2420 radio component handles packet transmission on TinyOS. Figure 3.1 shows the transmission and receiving process. When a node has a message to send, its micro-controller first transmits the message to the RAM (TXFIFO buffer) of the CC2420 radio component. After the buffering is done, CC2420 sends a signal to the micro-controller. At this time, if the radio channel is clear, the micro-controller signals CC2420 to send out the packet with a STXON strobe. Otherwise, it will backoff randomly and then test the channel again. After receiving a STXON signal, CC2420 first sends 12 symbol periods, with

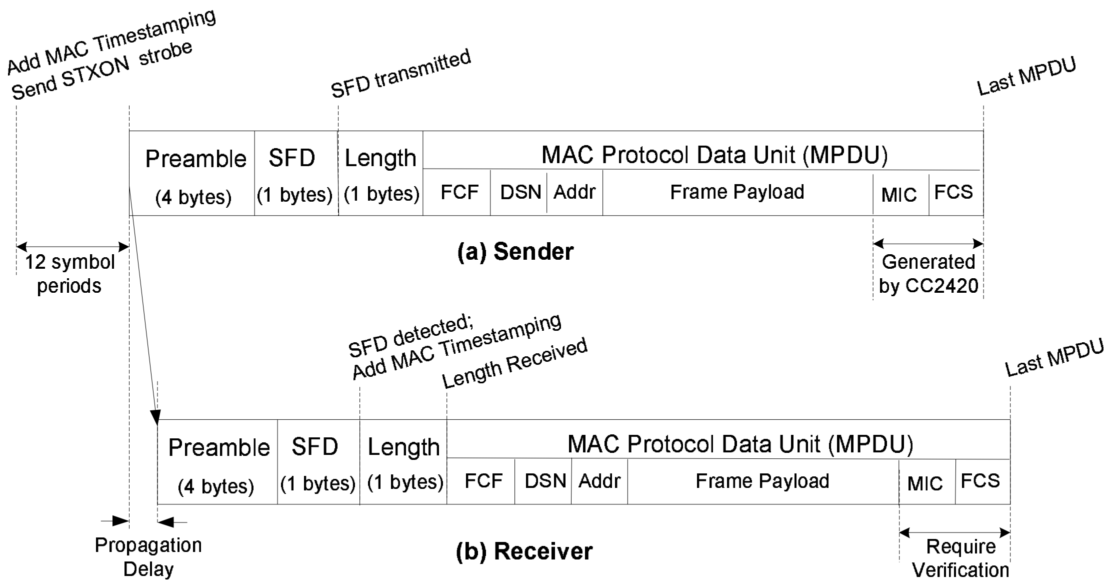


Figure 3.1: Packet sending and receiving process in pair-wise time synchronization

4 bits in each symbol, and then sends 4 byte preamble and 1 byte of Start of Frame Delimiter (SFD) field, followed by 1 byte length field and the MAC Protocol Data Unit (MPDU). The sequence of events follows strict timing, and the delays introduced by all of them are predictable.

We use the last bit of the SFD byte as the reference point for time synchronization. In other words, the sender takes the transmission (completion) time of the last bit of SFD as the MAC layer transmission timestamp, and the receiver marks the receiving time of the same bit as the receiving timestamp. To allow the sender to perform MAC layer timestamping and authentication, as mentioned earlier, we can predict the time when the last bit of SFD will be transmitted.

Sender Side: Now let us describe our proposed sending process in detail. Assume the sender has started sending a synchronization message to the RAM (TXFIFO buffer) of CC2420. At this time, the timestamp field in the message has not been filled. Upon completion of the transfer, CC2420 sends a signal to the micro-controller, which then starts handling the signal in the MAC layer. If the radio channel is clear, the micro-controller generates a timestamp by adding the current time with a constant offset Δ. This constant offset Δ is the time delay from checking the current time to the transmission of the last bit of SFD. The micro-controller then writes the timestamp directly to the corresponding bytes in CC2420's TXFIFO. Next, if the radio channel is still clear, it signals CC2420 to send out the message with a STXON strobe. Otherwise, it backs off for a random period of time and then repeats the above process. (Note that this backoff will force the micro-controller to write the MAC layer timestamp again when the same message is to be re-transmitted.) Upon receiving the STXON signal, as described earlier, CC2420 starts transmitting the symbol periods, the preamble, the SFD, and the MPDU. In the case when CC2420 can successfully transmit the packet, the execution and the data transmission are both deterministic, and the delay Δ is a constant. The delay Δ we obtained on MICAz motes is 399.28 μs. This includes the total transmission time for the 12 symbol periods, preamble and SFD ($(12{*}4{+}(4{+}1){*}8)/250,000 = 0.000352$ s $= 352\,\mu$s) and the execution time between checking the timestamp and starting the transmission ($47.28\ \mu$s).

In our implementation, we have CC2420 start the in-line authentication to generate the MIC of the message at the time when it begins to transmit the symbol periods. According to the manual of CC2420 [6], the in-line authentication component in CC2420 can generate a 12-byte MIC on a 98-byte message in 99 μs. Thus, we can easily see the MIC generation can be completed before it is transmitted. Besides the MIC, CC2420 also generates a 2-byte Frame Check Sequence (FCS) using Cyclic Redundancy Check (CRC).

Receiver Side: After an approximately 2 μs propagation delay [6], the radio component CC2420 on the receiver node will receive the preamble of an incoming message. Once the SFD field is completely received by CC2420, the SFD pin will go high to signal the micro-controller, which then records the current time as the receiving timestamp. When the FIFOP pin goes low, the micro-controller will be signaled to read the data from CC2420's RXFIFO buffer, in which the first byte indicates the length of the message. During the receiving process, CC2420 performs in-line verification of the MIC (and the CRC) in the message, using the pair-wise key shared between the sender and the receiver. The micro-controller examines the verification result, and copies the whole packet if the packet is authenticated. All these operations are performed in the MAC layer, and transparent to the application layer.

Unlike the deterministic delay on the sender's side, the delay affecting the receiving timestamps on the receiver's side is not entirely deterministic. When interrupt is disabled, the micro-controller will not be able to get the SFD signal immediately, and the resulting delay will be uncertain.

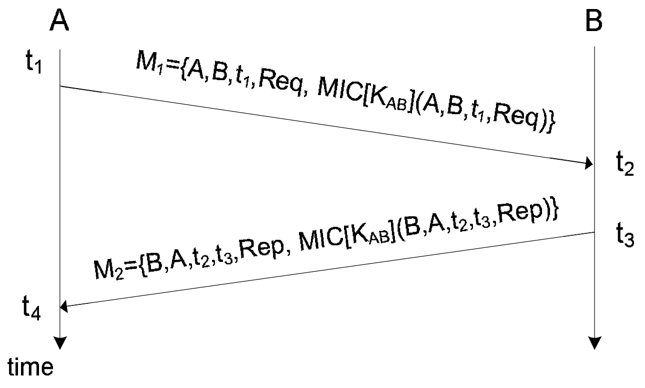

A
B
t_1
$M_1=\{A,B,t_1,Req, MIC[K_{AB}](A,B,t_1,Req)\}$
t_2
$M_2=\{B,A,t_2,t_3,Rep, MIC[K_{AB}](B,A,t_2,t_3,Rep)\}$
t_3
t_4
time

Figure 3.2: Revised secure single-hop pair-wise synchronization

3.2.2 Secure Single-Hop Pair-Wise Synchronization

Given the authenticated MAC layer timestamping capability, we can now describe how two neighbor nodes can perform secure pair-wise time synchronization.

Let us take a look at our options. RBS uses a receiver-receiver approach to synchronize nodes [26], in which a reference node broadcasts a reference packet to help pairs of receivers to identify their clock difference. However, an adversary can compromise it by simply launching a pulse-delay attack [29] or wormhole attack against one of the nodes to manipulate the packet transmission delay [44], so that the two nodes receive the reference packet at different times. In some protocols such as FTSP [63], one node passes its own time to the other by directly sending a MAC layer timestamped packet to the latter. This works well in benign environments, as demonstrated in [63]. However, in hostile environments, it suffers from the same problems mentioned above. TPSN uses a sender-receiver approach (through one request and one reply message) to help the sender obtain its clock difference from the receiver [31]. This approach was later improved with security in Secure Pair-Wise Synchronization (SPS) [29] to deal with pulse-delay and wormhole attacks. Specifically, it authenticates the messages being exchanged, and uses the timestamp information to estimate both the clock difference and the transmission delay. Pulse-delay and wormhole attacks that manipulate packet transmission delay will introduce extra delay in message transmission, and will be detected.

We adopt the SPS approach [29] with a slight modification. SPS uses a random nonce to prevent replay of a previously transmitted reply message. In our case, we simply use the sender's timestamp in the reply message to prevent replay attack, so that we can further reduce the message

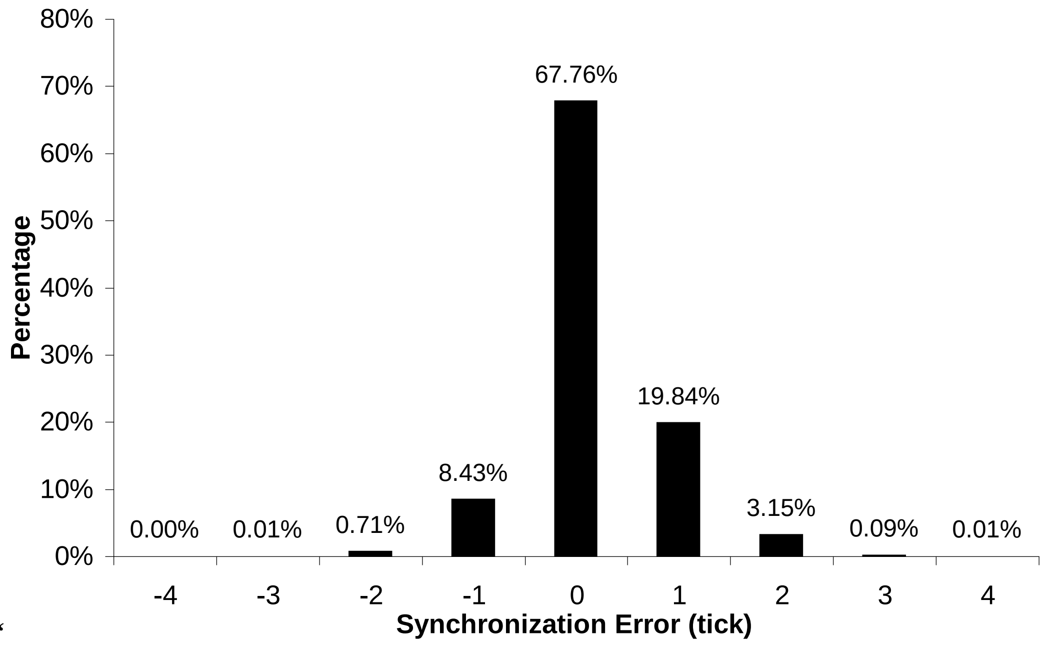


Figure 3.3: Distribution of synchronization error in pair-wise synchronization (1 tick = 8.68 μs.)

size.

Figure 3.2 shows the revised SPS protocol, in which all messages are timestamped and authenticated with the key K_{AB} shared by nodes A and B, as described in Section 3.2.1. Node A initiates the synchronization by sending message M_1. The message contains M_1's sending time t_1. Node B receives the message at t_2. After verifying the message, at time t_3, node B sends a message M_2 that includes t_2, t_3 to node A. When node A receives the message at t_4, it can calculate the clock difference $\Delta_{A,B} = \frac{(t_2-t_1)-(t_4-t_3)}{2}$, and the estimated one-way transmission delay $d_A = \frac{t_2-t_1+t_4-t_3}{2}$. Since all messages are authenticated, any modification to any message will be detected. To prevent the pulse-delay attacks [29] and wormhole attacks [44], node A verifies that the one-way transmission delay is less than the maximum expected delay. As a result, the sender A can easily detect attempts to affect the timeliness of the synchronization messages. Thus, our revised SPS achieves the same degree of security as the original one with smaller messages.

Note that the revised SPS protocol only enables the sender (A) to obtain the clock difference with the receiver (B). If the receiver (B) also needs this information, it has to initiate this protocol with the sender (A) as well. An alternative is to perform a three-way message exchange so that both nodes will get the clock difference at the end of the protocol execution. However, in

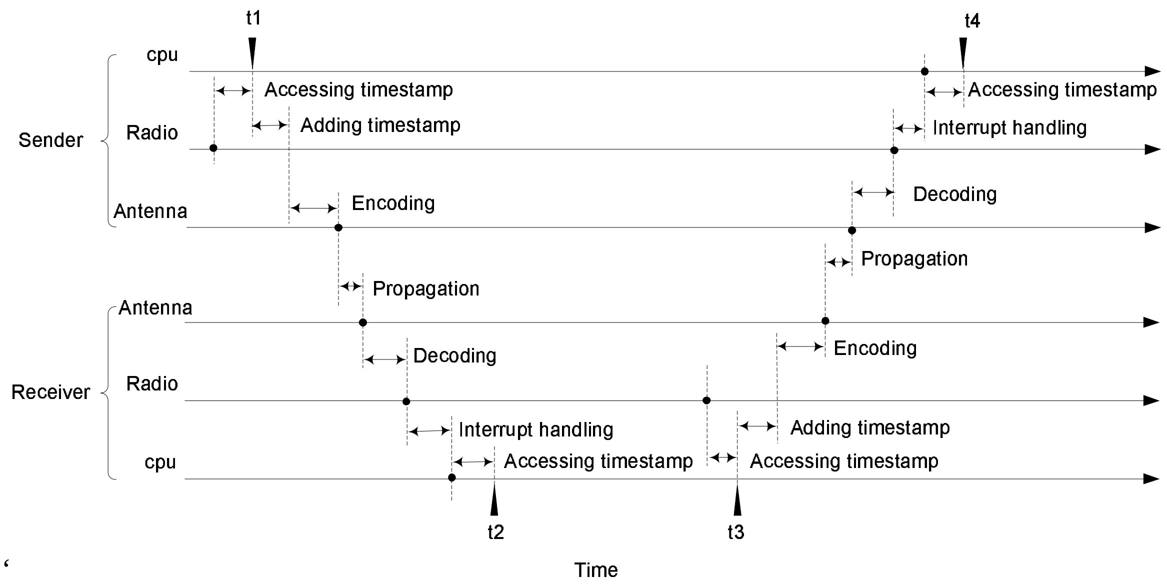


Figure 3.4: Delay uncertainty.

such a three-way protocol, both the sender and the receiver have to maintain their states at the intermediate protocol steps, and each node has to carefully maintain its states to avoid interference when it is involved in multiple concurrent synchronizations with different neighbors. The additional space requirement and the increased software complexity do not strictly justify the possibly reduced communication overhead.

We tested 30 pairs of nodes in our lab to obtain the synchronization precision. After two nodes finish a pair-wise time synchronization, a third reference node broadcasts a query to them. Each of the node records the MAC layer receiving time of the broadcast message and sends the receiving time to the reference node. This allows the reference node to calculate the synchronization error. Figure 3.3 shows the distribution of the pair-wise synchronization error. We present the sources of the message delivery delays in Figure 3.4. The time for the CPU to access current time is deterministic and less than 1 tick. The CPU can add the current time stamp into the buffered message in less than 2 ticks. The time for adding timestamp is deterministic too. The propagation time depends on the distance between the sender and the receiver, and it is highly deterministic. The uncertainty is mainly because of encoding/decoding times and the jitter of the interrupt handling. The encoding time is for the radio to encode and transform a part of the message to electromagnetic waves; the decoding time is for the radio to transform and decode the message from electromagnetic waves to binary data. Note, the encoding and decoding times may not be the same on the sender and the receiver. When encoding, each byte is divided into two symbols, 4 bits each. Then, each symbol is mapped to one out of 16 pseudo-random sequences, 32 chips each. The chip sequence is transmitted at 2 MChips/s, with the least significant chip transmitted first for each symbol. The receiver performs the reverse process when decoding. The jitter of interrupt handling is caused by the disabled interrupt on the message receiving side. The error span can be further reduced by having a smaller tick size; however, this will also increase the overhead in maintaining the clock ticks through interrupt handling.

3.2.3 Security Analysis

The secure single-hop pair-wise time synchronization uses hardware-assisted in-line authentication, providing authentication of the source and the content of synchronization messages. Moreover, it uses a two-way message exchange to estimate both the clock difference between direct neighbors and the transmission delay, and can detect attacks that attempt to mislead time synchronization by introducing extra message delays. Thus, it provides protection of the source, the content, and the timeliness of single-hop pair-wise synchronization messages. Specifically, this technique effectively defeats external attacks that attempt to mislead single-hop pair-wise time synchronization, including forged and modified messages, pulse-delay attacks, and wormhole attacks that introduce extra delays. This technique cannot handle DoS attacks that completely jam the communication channel. Nevertheless, no existing protocol can survive such extreme DoS attacks.

3.3 Implementation Details

Our implementation is targeted at MICAz motes [4]. (However, our implementation can be used with slight modification for other sensor platforms that also use CC2420 radio components, such as TelosB [7] and Tmote Sky [8].) MICAz has an 8-bit micro-controller *ATMega128L* [1], which has 128 kB program memory and 4 kB SRAM. As discussed earlier, MICAz is equipped with the ChipCon CC2420 radio component [6], which works at 2.4GHz radio frequency and provides up to 250 kbps data rate. CC2420 is an IEEE 802.15.4 compliant RF transceiver that features hardware security support.

In the following, we give a few details that are critical for repeating our implementation.

3.3.1 Exploiting Hardware Security Support in CC2420

The hardware security support featured by CC2420 provides two types of security operations: *stand-alone encryption operation* and *in-line security operation*. The stand-alone encryption operation provides a plain AES encryption, with 128 bit plain-text and 128 bit keys. To encrypt a plain-text, a node first writes the plain-text to the stand-alone buffer *SABUF*, and then issues a SAES command to initiate the encryption operation. When the encryption is complete, the cipher-text is written back to the stand-alone buffer, overwriting the plain-text.

The in-line security operation can provide encryption, decryption, and authentication on frames within the receive buffer (RXFIFO) and the transmit buffer (TXFIFO) of CC2420 on a frame basis. It supports three modes of security: *counter mode (CTR)*, *CBC-MIC*, and *CCM*. CTR mode performs encryption on the outgoing MAC frames in the TXFIFO buffer, and performs decryption on the incoming MAC frames in the RXFIFO buffer. CBC-MIC mode can generate and verify the message integrity code (MIC) of the messages. The length of MIC can be adjusted. CCM mode combines CTR mode encryption and CBC-MIC authentication in one operation. All the three security modes are based on AES encryption/decryption using 128 bit keys.

We use the CBC-MIC mode to authenticate both pair-wise and global synchronization messages. A sender can use in-line CBC-MIC mode to generate the MIC for both pair-wise and global synchronization messages in the MAC layer after the message has been written to the TXFIFO buffer.

The receiver side, however, is slightly different. When a receiver receives a pair-wise synchronization message, since it already knows the secrete key shared with the sender, it can use the in-line CBC-MIC mode to verify the MIC before the message is read from the RXFIFO buffer. However, for the global synchronization messages, before receiving the disclosed key, the receiver cannot use in-line authentication to verify the MIC in the message. Because the receiver still needs the RXFIFO buffer to receive other messages, it cannot buffer the message in the RXFIFO buffer while waiting for the disclosed key. Thus, we have the receiver read the message from RXFIFO and buffer it in its local memory. When the key is received, the receive uses the stand-alone mode to authenticate the buffered global synchronization messages. Since the stand-alone mode only provides single-block encryption functionality, we implemented the CBC mode based on the hardware support.

3.3.2 Handling Timers

Using timers on MICAz is a tricky issue; improper uses usually lead to unexpected results. The micro-controller ATMega128 provides two 8-bit timers (Timer0, Timer2) and two 16-bit timers (Timer1, Timer3) [1]. In TinyOS, Timer 0 is mainly used as one-shot or repeat timers for applications. For MICAz, Timer 2 is used by CC2420 as a high precision timer (32 μs per tick) to backoff the sending packets for a short period of time. Timer 1 is used by CC2420 for capturing radio packet transmit and receive events. In our implementation, we use the remaining 16-bit Timer 3 to maintain the local clock and schedule the message transmission.

ATMega128L uses a 7.3728 MHz crystal oscillator as I/O clock source, whose accuracy is $\pm 40ppm$ [6]. In our implementation, we divide the I/O clock by 64 as the source of Timer 3, thereby achieving a 115.2 kHz Timer 3, with a 8.68 μs time resolution. Timer3 provides three compare match registers (*OCR3A/B/C*), each connected with an interrupt vector. If the compare match interrupt is enabled, whenever the value of Timer3 (TCNT3) equals to the value of one compare match register, it will trigger an interrupt to handle the event. Each node uses compare match register A to maintain a 48-bit logical clock. The value of Timer3 (TCNT3) is 16 bits, and it will overflow every 568.8 ms. We add another 32 bits to have a logical clock that will not overflow for over 77 years Each node sets compare match register B to launch pair-wise synchronization with its neighbors periodically. The source node will use compare match register B to initiate the global synchronization periodically. Each node uses compare match register C to send its global synchronization message in its nearest short μTESLA interval and disclose the key in the adjacent long μTESLA interval.

3.4 Summary

In this chapter, we presented a secure single-hop pair-wise time synchronization technique based on hardware-assisted, authenticated MAC layer timestamping. This technique exceeds the capability of previous solutions. In particular, unlike the previous attempts, our technique can handle high data rate such as those produced by MICAz motes (in contrast to those by MICA2 motes). This technique provides protection of the source, the content, and the timeliness of single-hop pair-wise synchronization messages. It can effectively defeat external attacks that attempt to mislead single-hop pair-wise time synchronization. We implement the proposed techniques on MICAz motes running TinyOS. The secure single-hop pair-wise time synchronization serves as the building block to achieve the secure and resilient global time synchronization in Chapter 5 and Chapter 6.

Chapter 4

Fault-Tolerant Cluster-Wise Time Synchronization

In wireless sensor networks, it is usually necessary to have a cluster of nodes share a common view of a local clock time, so that the nodes can coordinate their actions. For example, in time slotted MAC protocols, the multiple access to the shared communication medium is achieved by assigning time slots to a group of nodes. Sensor nodes need to have a synchronized clock to access their time slots without colliding with other nodes. As another example, to increase the energy efficiency, a cluster of sensor nodes may frequently switch into power-saving sleep mode at the same time [100]. They also need a common clock to coordinate their sleep/listen periods. In benign environments, such a local common clock can be easily achieved by having all the nodes synchronize to a given node. However, in hostile environments where some nodes may be compromised, it is quite challenging to synchronize the clocks among a cluster of nodes. Indeed, none of the aforementioned time synchronization protocols can survive malicious actions by compromised nodes. A compromised node may disrupt the time synchronization by sending different time to non-compromised nodes. For example, when RBS [26] is used for pair-wise synchronization, a compromised node may provide different non-compromised nodes different time values about the receipt of the reference packet.

We develop a novel fault-tolerant cluster-wise clock synchronization scheme for clusters of sensor nodes, where the nodes in each cluster can communicate with each other directly through broadcast [92]. In each round of time synchronization, only one node serves as the *synchronizer*, and only one authenticated synchronization message is broadcast. Thus, our scheme can avoid the message collision problem in the previous schemes. The proposed scheme exploits a recently proposed local broadcast authentication technique for sensor networks, which is purely based on symmetric cryptography [104], thus avoiding the costly digital signature for message authentication. Our analysis shows that the proposed scheme guarantees an upper bound on the clock difference between nonfaulty nodes when no more than $1/3$ of the nodes are compromised and collude with each other. In Section 4.3, we propose a secure distributed cluster formation algorithm which can divide a whole sensor network into multiple mutual disjoint cliques [91]. Then, we can run our fault-tolerant cluster-wise clock synchronization scheme in each clique.

4.1 Cluster-Wise Time Synchronization Model

In this section, we describe our model for fault-tolerant clock synchronization in sensor networks, which is adapted from [22]. For readers' convenience, Table 4.1 lists the notations used in this chapter.

Table 4.1: Notations in fault-tolerant cluster-wise time synchronization

n	The number of nodes in a cluster
m	The number of colluding malicious nodes in a cluster
$C_i(t)$	Clock time at node i when the real time is t
ρ	Maximum drift rate of all well-behaved clocks
ψ	Maximum message transmission delay between two neighboring nodes
ϵ	Maximum clock reading error
R	Synchronization interval
beg^f	The real time at which the first nonfaulty node starts its f-th logical clock
end^f	The real time at which the last nonfaulty node starts its f-th logical clock
δ	Maximum clock drift over $[end^f, end^{f+1}]$ for any f

Sensor nodes usually contain inexpensive crystal oscillators, and the typical clock drift rate is tens of microseconds [81]. A clock C is considered *well-behaved* if its rate of drift from the real time is bounded by a constant $\rho > 0$ for all the real time points t_1 and t_2, where $t_1 < t_2$. The rate of drift between any two well-behaved clocks is bounded by $\lambda = \rho(2 + \rho)/(1 + \rho)$, which is less than 2ρ.

A sensor node is *nonfaulty* if it correctly executes a given time synchronization algorithm and its clock is well-behaved; otherwise, it is a faulty node. We assume that clocks are synchronized in rounds, each of which consists of R time units. We denote the real time point at which the first (or the last) nonfaulty node starts its f-th round as beg^f (or end^f). Over the time interval $[end^f, end^{f+1}]$ for any f, there exists a maximum clock drift δ between any two well-behaved clocks, i.e.,

$$\delta = 2\rho(end^{f+1} - end^f). \tag{4.1}$$

Suppose a node makes a clock adjustment at time t. We use $C(t)$ and $C^+(t)$ to represent the clock time before and after the clock adjustment, respectively. Suppose there is an upper bound ψ for a message to be sent by a node, transmitted, and processed by the recipients of the message. Suppose node i sends a message at $C_i(t_1)$, node j receives the message at $C_j(t_2)$, where $0 < t_2 - t_1 < \psi$, and node j adjusts its clock to $C_j^+(t_2) = C_i(t_1)$. Then,

$$C_i(t_2) - C_j^+(t_2) = C_i(t_2) - C_i(t_1) < \epsilon, \tag{4.2}$$

where $\epsilon = (1+\rho)\psi$ is the upper bound for the clock reading error, which includes the maximum transmission delay and the clock drift during this delay. We assume at the "starting time" t_0, the clock difference between two nonfaulty nodes i and j is less than δ_0, i.e.,

$$|C_i(t_0) - C_j(t_0)| < \delta_0. \tag{4.3}$$

In the next section, we develop a new fault-tolerant cluster-wise time synchronization scheme for sensor networks. Suppose there exist up to $m < \frac{n}{3}$ malicious nodes in a cluster of n nodes that can communicate with each other through broadcast. With $k = \frac{n - m\frac{\epsilon}{(\delta+\epsilon)}}{n-3m}$ and $\Delta = \delta + \epsilon(1+4\rho)$, our algorithm satisfies the following two conditions:

- *CS1:* For any two nonfaulty nodes i and j, there exists an upper bound on the clock difference between them for any real time point. That is, for all $f \geq 1$, and $t \in [beg^f, beg^{f+1}]$, $|C_i(t) - C_j(t)| \leq (2km+1)\Delta + m\delta + 2\rho\epsilon$;

- *CS2:* If a node makes an adjustment to its clock at time t, there is an upper bound on the clock adjustment. That is, $|C^+(t) - C(t)| \leq k\Delta$.

4.2 Fault-Tolerant Cluster-Wise Time Synchronization

4.2.1 Overview

In this chapter, we focus on providing fault-tolerant clock synchronization within a cluster of nodes, where a message broadcast by one node can reach all the other nodes in the cluster. We assume that each node has a unique ID, and every two nodes in the cluster share a unique pair-wise key. (Such pair-wise keys can be provided by several key predistribution schemes proposed recently [56, 17, 25].) One node can obtain a unique ID by manual assignment, or derive it from its physical characteristics. One node can identify another node using the unique pair-wise key they share. A potential threat against this assumption is Sybil attacks [24], where a malicious node impersonates multiple nodes by claiming multiple IDs. Fortunately, recent studies [69] show that the aforementioned key predistribution schemes can reduce the probability that an attacker can fabricate new IDs close to zero even if a fair number of nodes are compromised. An attacker may certainly increase this probability by compromising a large number of nodes. However, in such cases, the whole key predistribution scheme is also compromised, and as a result, there is no security in such networks.

Our fault-tolerant time synchronization scheme executes once for every R time units. For convenience, we call such a R time unit period a *round*. In each round, one node in the cluster serves as the *synchronizer*, which broadcasts a *synchronization message* to the other nodes; all the other nodes then synchronize their clocks accordingly.

We assume the clocks of the sensor nodes are synchronized initially. Moreover, we assume the nodes in a cluster agree on the order in which they serve as the synchronizer. We refer to this order as the *synchronizer order*. There are several ways to meet these two assumptions. For example, we may use the approach in [59] to achieve initial time synchronization, and adapt algorithm *OM* [54] to decide the synchronizer order in a cluster[1].

A practical method to meet the aforementioned assumptions is to add a bootstrapping phase during the deployment of a sensor network. We may use one or multiple trusted external

[1]Algorithm *OM* guarantees a group of $n-1$ nodes agree on a value sent from another node when there are at most $m < \frac{n}{3}$ malicious nodes [54]. A cluster of n nodes may execute algorithm OM n times to guarantee that all the nonfaulty nodes obtain one value from each node. Each node can then use, for example, the XOR of all the values as a seed to generate the synchronizer order. To prevent malicious nodes from manipulating the synchronizer order, we may use algorithm OM n times to first distribute a set of commitments (e.g., hash images) of these values, and then execute it for another n times to distribute the original values. Though this approach can be used to decide the synchronizer order in a cluster in a fault tolerant way, it is not scalable and thus not preferred in practice.

devices, which maintain well synchronized clocks (e.g., through GPS receivers), to facilitate the bootstrap of the sensor network. It is normally reasonable to assume that the sensor nodes are not compromised during the deployment of the network. Thus, the external devices can distribute synchronized initial clock values to all the sensor nodes. At the same time, the external devices can collect neighbor information from the sensor nodes, form clusters among them, and distribute the synchronizer order in each cluster. If security is of concern during the bootstrapping phase, a symmetric key may be shared between each sensor node and trusted device. The entire bootstrapping phase can be fully automated, and performed while a sensor network is being deployed. There are certainly other feasible ways to meet the same assumptions.

In the proposed algorithm, each node maintains a counter f, initialized as 1 and incremented by 1 in each round. Suppose a node's clock time reaches $f \times R$ time units, where R is the number of time units in each round. If this node is the synchronizer, it immediately broadcasts an authenticated message to all the other nodes. When a non-synchronizer node receives such a synchronization message, it examines the message. If the message is invalid or the sender is not the designated synchronizer, the receiver simply drops the message. Otherwise, the receiver adjusts its clock according to the time when the synchronization message is received. (Note that the receiver can determine that the synchronizer's clock must be $f \times R$ time units after the start of time synchronization.)

Our scheme works under the *arbitrary attack model* [28], in which malicious nodes can arbitrarily deviate from the protocol (e.g., sending conflicting messages to different nodes with directional antenna) and collude with each other. Because communication failures can be considered as sending node failures, we do not consider it separately. We assume an attacker may replace a compromised sensor node with a resourceful node (e.g., a laptop with directional antenna), thus gaining advantage over the regular nodes. Since we only care about the clock difference between nonfaulty nodes, for brevity, we will use "the maximum clock difference" to mean "the maximum clock difference between any two nonfaulty nodes".

In the following, we first discuss the authentication of the broadcast synchronization messages, then describe and analyze the proposed scheme, and finally compare the proposed scheme with several traditional fault-tolerant time synchronization schemes.

4.2.2 Local Broadcast Authentication

In each round of time synchronization, only one node serves as the synchronizer and broadcasts a synchronization message. To prevent malicious nodes from impersonating nonfaulty synchronizers, each synchronization message must be authenticated.

The proposed scheme does not require a clock value be sent in a synchronization message. After receiving a synchronization message from the synchronizer, a node knows how to adjust its clock. Thus, a receiving node only needs to verify that a message is sent from the correct synchronizer and the message is not replayed by malicious nodes.

We adapt a recently proposed local broadcast authentication scheme for sensor networks [104] to authenticate the broadcast synchronization messages. At the beginning, each node generates a one-way key chain $\{K^{(0)}, K^{(1)}, ..., K^{(l)}\}$ in the following way: $K^{(i-1)} = F(K^{(i)}), (1 \leq i \leq l)$, where $K^{(l)}$ is a random number, and F is a one-way function. Each node sends $K^{(0)}$ as the *commitment* of its key chain to other nodes, authenticated with the shared pair-wise keys with those nodes. The keys in the key chain are disclosed in the reverse order to their generation. When a node serves as the synchronizer, it appends the next undisclosed key in the key chain to the broadcast message. When the other nodes receive the message, they verify that the message is sent from the claimed node using the commitment or the recently disclosed key of the node. Note that $K^{(i)} = F^{i-j}(K^{(j)})$ when $i > j$. Thus, even if a node fails to receive all the keys between $K^{(j)}$ and $K^{(i)}$ from a given synchronizer, it still can verify the key $K^{(i)}$ with $K^{(j)}$. Due to the property of one-way function, a malicious node cannot know an undisclosed key belonging to a nonfaulty node. Each node only accepts the first copy of a broadcast message, and drops the duplicated ones. Therefore, a malicious node cannot forge or reuse nonfaulty nodes' broadcast messages. An attacker may certainly shield some victim nodes from receiving the first copy of the synchronization message, or create a wormhole [44] between nonfaulty nodes. As a result, the victim node may accept a delayed synchronization message. Such attacks are equivalent to having a malicious node as the synchronizer, and can be handled when the total number of malicious or shielded nonfaulty synchronizers is no more than $m < \frac{n}{3}$. This broadcast authentication scheme needs n^2 unicast messages to exchange the commitments of all the nodes' key chains during the initialization phase.

4.2.3 Fault-Tolerant Cluster-Wise Time Synchronization Algorithm

The proposed scheme executes one round of time synchronization every R time units. For simplicity, we assume the "starting time" is $beg^0 = end^0 = 0$. For any two nonfaulty nodes i and j, $|C_i(0) - C_j(0)| < \epsilon(1 + 4\rho)$. Each node maintains a counter f by increasing it by one in each round of time synchronization. Initially $f = 1$. We assume each node i has generated a one-way key chain, and exchanged the commitment $K_i^{(0)}$ with the other nodes.

The algorithm consists of two tasks that run continuously on each nonfaulty sensor node. In the first task, if node i is the synchronizer for the f-th round of synchronization, when its clock time reaches $C = f \times R$, it immediately broadcasts a synchronization message "$N_i|K_i^{(\lceil f/n \rceil)}$" to all the other nodes, where N_i is node i's ID and $K_i^{(\lceil f/n \rceil)}$ is the key in node i's key chain that is used for authentication in the f-th round.

In the second task, when a node receives a synchronization message at its clock time T in the f-th round of time synchronization, if $T < f \times R - x$ or $T > f \times R + x$, the node drops the message. In our algorithm, $x = (2km + 1)\Delta + m\delta$ is the maximum clock difference between any nonfaulty node and a nonfaulty synchronizer, where m is the number of malicious nodes, $k = \frac{n - m\frac{\epsilon}{\delta+\epsilon}}{n-3m}$, and $\Delta = \delta + \epsilon(1+4\rho)$ is the maximum clock difference between any two nodes if all the nodes are nonfaulty. Otherwise, it verifies that node N_i is the correct synchronizer and it is the first time to receive the $K_i^{(\lceil f/n \rceil)}$ and $F(K_i^{(\lceil f/n \rceil)}) = K_i^{(\lceil f/n \rceil - 1)}$, where F is the one-way function and $K_i^{(\lceil f/n \rceil - 1)}$ is the key received from node i in the $(f - n)$-th round or the commitment. If the message cannot pass these verifications, the node drops the message. Otherwise, the node calculates the clock difference $\bar{\Delta} = f \times R - T$ and performs the following clock adjustment: if $|\bar{\Delta}| < k\Delta$, the node adjusts its clock time by adding $\bar{\Delta}$; if $k\Delta \leq \bar{\Delta} \leq x$, it increases its clock time by $k\Delta$; if $-x \leq \bar{\Delta} \leq -k\Delta$, it decreases its clock time by $k\Delta$. The node also increments the counter f by 1. If the node does not receive an authenticated synchronization message for the current round by the time $f \times R + x$, it increments the counter f by 1 and enters the next round. Our algorithm guarantees that the synchronized nodes maintain the same counter f after each round of time synchronization.

A node may lose synchronization from a cluster, for example, due to long-term communication failures. If this failure node is able to re-establish direct and secure communication with the other nodes in the same cluster, it may attempt to recover from such a failure. One possible approach is to request the current clock values from all the other nodes and then determine the local clock value by choosing the median. (Note that this node can easily determine the counter value f

using the recovered clock value and the synchronization interval R.) If the majority of these nodes are non-faulty and have been maintaining synchronized clocks, then there must exist two non-faulty nodes n_1 and n_2 whose clock values are T_1 and T_2, respectively, such that the above median clock value is between T_1 and T_2. In other words, the failure node can successfully set its local clock to a value in the acceptable range. However, in other cases, the failure node is not guaranteed to recover.

Algorithm 4.1 shows the pseudo code. Because all the nodes serve as the synchronizer in a round robin fashion, we refer to our scheme as *Synchronizer Ring (SR)* algorithm. To ensure that clocks are never set back, we may further adapt the technique proposed in [53], which spreads each synchronization adjustment over the next synchronization period. Due to the space limit, we omit the details. In the following, we first examine the proposed technique when there is no malicious participant, and then investigate it when there are colluding attacks from compromised nodes.

Lemma 4.2.1 *After a nonfaulty node i adjusts its clock to a nonfaulty synchronizer s's clock at t_i^f, where $beg^f \leq t_i^f \leq end^f$, for any $t \in [t_i^f, end^f]$, $-2\rho\epsilon < C_s(t) - C_i(t) < \epsilon(1 + 2\rho)$.*

Proof: For the right part, by inequality 4.2, we have $C_s(t) - C_i(t) < |C_s(t_i^f) - C_i^+(t_i^f)| + 2\rho(t - t_i^f) < \epsilon + 2\rho(end^f - beg^f) < \epsilon(1+2\rho)$. For the left part, by inequality 2.1, we have $(C_i(t) - C_s(t)) - (C_i^+(t_i^f) - C_s(t_i^f)) \leq |(C_i(t) - C_i^+(t_i^f)) - (C_s(t) - C_s(t_i^f))| < 2\rho(t - t_i^f) < 2\rho\epsilon$. By inequality 4.2, we have $-\epsilon < C_i^+(t_i^f) - C_s(t_i^f) < 0$. Thus, $C_i(t) - C_s(t) < 2\rho\epsilon + (C_i^+(t_i^f) - C_s(t_i^f)) < 2\rho\epsilon$. Together, we have $-2\rho\epsilon < C_s(t) - C_i(t) < \epsilon(1 + 2\rho)$.

Theorem 4.2.2 *Suppose for any two nodes i and j, $|C_i(end^0) - C_j(end^0)| < \epsilon(1+4\rho)$. If all nodes are nonfaulty, Algorithm SR is executed, and there is no communication failure, $|C_i(t) - C_j(t)| < \delta + \epsilon(1 + 6\rho)$ for all $t > end^0$.*

Proof: First, we prove by induction that for all $f \geq 0$, $|C_i(end^f) - C_j(end^f)| < \epsilon(1 + 4\rho)$. From the assumption, we have $|C_i(end^0) - C_j(end^0)| < \epsilon(1 + 4\rho)$. Suppose at time point end^f, $|C_i(end^f) - C_j(end^f)| < \epsilon(1 + 4\rho)$, we need to prove that at time point end^{f+1}, $|C_i(end^{f+1}) - C_j(end^{f+1})| < \epsilon(1 + 4\rho)$.

Suppose the $(f + 1)$-th synchronizer is s. Since there is no communication failure, further assume nodes i and j adjust their clock times at t_i^{f+1} and t_j^{f+1}, respectively, where $beg^{f+1} \leq t_i^{f+1} \leq t_j^{f+1} \leq end^{f+1}$. We consider three time intervals separated by t_i^{f+1} and t_j^{f+1} in $[end^f, end^{f+1}]$.

Algorithm 4.1 Synchronizer Ring

```
Initialization
$f \leftarrow 1$; $k \leftarrow \frac{n-m\frac{\epsilon}{\delta+\epsilon}}{n-3m}$; $\Delta = \delta + \epsilon(1+4\rho)$; $x \leftarrow (2km+1)\Delta + m\delta$;
Task 1: Send
if ($C = f \times R$) and ($Order(N_i) = f \; mod \; n$) then
  Broadcast a message "$N_i|K_i^{(\lceil f/n \rceil)}$";
end if
Task 2: Receive
if (Receive a message "$N_i|K_i^{(\lceil f/n \rceil)}$" at $T$) then
  if ($f \times R - x \leq T \leq f \times R + x$) and ( $F(K_i^{(\lceil f/n \rceil)}) = K_i^{(\lceil f/n \rceil - 1)}$) and ($Order(N_i) = f \; mod \; n$)
  then
    $\bar{\Delta} \leftarrow f \times R - T$;
    if $k\Delta \leq \bar{\Delta} \leq x$ then
      $\bar{\Delta} \leftarrow k\Delta$;
    else if $-x \leq \bar{\Delta} \leq -k\Delta$ then
      $\bar{\Delta} \leftarrow -k\Delta$;
    end if
    $C \leftarrow C + \bar{\Delta}$; $f \leftarrow f + 1$;
  else
    Drop the message;
  end if
end if
if Has not received a correct synchronization message by $f \times R + x$ (Note that this may be implemented
as a timer.) then
  $f \leftarrow f + 1$;
end if
```

For any $t \in [end^f, t_i^{f+1})$, by inequalities 2.1 and 4.1, we have $|C_i(t)-C_j(t)| < |C_i(end^f)-C_j(end^f)| + 2\rho(t - end^f) < \epsilon(1+4\rho)+\delta$.

For any $t \in [t_i^{f+1}, t_j^{f+1})$, by Lemma 4.2.1, we have $-2\rho\epsilon < C_s(t) - C_i(t) < \epsilon(1+2\rho)$. For node j, if $C_s(t) > C_j(t)$, we have $0 < C_s(t) - C_j(t) < |C_s(end^f) - C_j(end^f)| + 2\rho(t - end^f) < \epsilon(1+4\rho)+\delta$. Thus, we have $|C_i(t) - C_j(t)| < \delta + \epsilon(1+6\rho)$. If $C_s(t) < C_j(t)$, we have $0 < C_j(t) - C_s(t) < \delta + 2\rho\epsilon$, and then $|C_i(t) - C_j(t)| < \delta + \epsilon(1+4\rho)$. Considering both cases, we have $|C_i(t) - C_j(t)| < \delta + \epsilon(1+6\rho)$.

For any $t \in [t_j^{f+1}, end^{f+1}]$, by Lemma 4.2.1, we have $-2\rho\epsilon < C_s(t)-C_j(t) < \epsilon(1+2\rho)$, and $-2\rho\epsilon < C_s(t) - C_i(t) < \epsilon(1+2\rho)$. Therefore, we have $|C_i(t) - C_j(t)| < \epsilon(1+4\rho)$. In particular, $|C_i(end^{f+1}) - C_j(end^{f+1})| < \epsilon(1+4\rho)$. Thus, $|C_i(end^f) - C_j(end^f)| < \epsilon(1+4\rho)$ for all $f \geq 0$.

According to the above proof, we can see that for all $f \geq 0$ and any $t \in [end^f, end^{f+1}]$, $|C_i(t)-C_j(t)| < \delta+\epsilon(1+6\rho)$. Thus, the inequality holds for any $t > end^0$ as long as the algorithm is executed.

The maximum clock difference $\delta + \epsilon(1+6\rho)$ can only be reached between two non-synchronizer nodes i and j during $[beg^{f+1}, end^{f+1}]$. During $[end^f, end^{f+1}]$, the clock difference between node i (or node j) and the synchronizer s is at most $\Delta = \delta + \epsilon(1+4\rho)$, which is the allowable maximum clock adjustment when all nodes are nonfaulty.

Now let us consider the cases where there are colluding malicious synchronizers.

Lemma 4.2.3 *If the f-th ($f \geq 1$) synchronizer is malicious, it can increase the maximum clock difference by at most $2k\Delta + \delta$ during $[beg^f, beg^{f+1}]$.*

Proof: According to Algorithm 4.1, a nonfaulty node adjusts its clock by at most $k\Delta$ in one round. Thus, over $[beg^f, end^f]$, a malicious synchronizer can increase one nonfaulty node's clock time by at most $k\Delta$, while decrease another nonfaulty node's clock time by at most $k\Delta$. (The malicious synchronizer may use directional antenna to launch such attacks.) Moreover, over the time interval $[beg^f, beg^{f+1}]$, the maximum clock drift is δ. In total, one malicious synchronizer can increase the maximum clock difference by at most $2k\Delta + \delta$.

Lemma 4.2.4 *Suppose two nonfaulty nodes i and j synchronize to a nonfaulty synchronizer s at t_i^f and t_j^f, respectively, where $beg^f \leq t_i^f \leq t_j^f \leq end^f$. If $|C_s(t_i^f) - C_i(t_i^f)| < k\Delta$ and $|C_s(t_j^f) -$*

$C_j(t_j^f)| < k\Delta$, then $|C_i(end^f) - C_j(end^f)| < \epsilon(1+4\rho)$.

Proof: According to Algorithm 4.1, because $|C_s(t_i^f) - C_i(t_i^f)| < k\Delta$, nodes i adjust their clocks to the synchronizer's clock at t_i^f. By Lemma 4.2.1, when $t = end^f$, we have $-2\rho\epsilon < C_s(end^f) - C_i(end^f) < \epsilon(1+2\rho)$. For node j, we have a similar result, i.e., $-2\rho\epsilon < C_s(end^f) - C_j(end^f) < \epsilon(1+2\rho)$. Thus, we have $|C_i(end^f) - C_j(end^f)| < \epsilon(1+4\rho)$.

Lemma 4.2.5 *Suppose during $[beg^{f+1}, end^{f+1}]$, the maximum clock difference $\bar{\Delta}$ is between node i and node j. If $\bar{\Delta} \le (2km+1)\Delta + m\delta$ and the $(f+1)$-th synchronizer is nonfaulty, for any $t \in [end^{f+1}, beg^{f+2}]$, $|C_i(t) - C_j(t)| \le MAX(\bar{\Delta} - (k-1)\Delta, \Delta)$.*

Proof: Suppose node i and j adjust clocks at time t_i and t_j. If $\bar{\Delta} < k\Delta$, by Lemma 4.2.4, for any $t \in [end^{f+1}, beg^{f+2}]$, $|C_i(t) - C_j(t)| < |C_i(end^{f+1}) - C_j(end^{f+1})| + 2\rho(t - end^{f+1}) < \epsilon(1+4\rho) + \delta = \Delta$.

When $k\Delta \le \bar{\Delta} \le (2km+1)\Delta + m\delta$, if node i is the synchronizer, according to our algorithm, node j adjusts its clock with $k\Delta$ at t_j. We have $|C_i(end^{f+1}) - C_j(end^{f+1})| < |C_i(t_j) - C_j^+(t_j)| + 2\rho(end^{f+1} - t_j) < (\bar{\Delta} - k\Delta + \epsilon) + 2\rho\epsilon$. So for any $t \in [end^{f+1}, beg^{f+2}]$, by inequality 4.1, we have $|C_i(t) - C_j(t)| < |C_i(end^{f+1}) - C_j(end^{f+1})| + 2\rho(t - end^{f+1}) < \bar{\Delta} - k\Delta + \epsilon(1+2\rho) + \delta < \bar{\Delta} - (k-1)\Delta$. When node j serves as the synchronizer, we have the same result.

If neither node i nor node j is the synchronizer, because the maximum clock difference is between i and j, the nonfaulty synchronizer s's clock time must be between these two nodes' clock times. If $|C_i(t_i) - C_s(t_i)| \ge k\Delta$ or $|C_j(t_j) - C_s(t_j)| \ge k\Delta$, node i or j adjust clocks by $k\Delta$, for any $t \in [end^{f+1}, beg^{f+2}]$, we have $|C_i(t) - C_j(t)| < \bar{\Delta} - (k-1)\Delta$. If $|C_i(t_i) - C_s(t_i)| < k\Delta$ and $|C_j(t_j) - C_s(t_j)| < k\Delta$, by Lemma 4.2.4, for any $t \in [end^{f+1}, beg^{f+2}]$, we have $|C_i(t) - C_j(t)| < \delta + \epsilon(1+4\rho) = \Delta$.

So for any $t \in [end^{f+1}, beg^{f+2}]$, $|C_i(t) - C_j(t)| \le MAX(\bar{\Delta} - (k-1)\Delta, \Delta)$.

Lemma 4.2.6 *When $n > 3m$, Algorithm 4.1 satisfies the following conditions: (1) For all $f \ge 1$ and $t \in [beg^f, beg^{f+1}]$, given any two nonfaulty nodes i and j, $|C_i(t) - C_j(t)| \le (2km+1)\Delta + m\delta + 2\rho\epsilon$; (2) If a node makes an adjustment to its clock at time t, then $|C^+(t) - C(t)| \le k\Delta$.*

Proof: Condition 2 is easy to prove, since $\bar{\Delta}$ is no greater than $k\Delta$ according to Algorithm 4.1. Now we prove Condition 1 by induction.

By inequalities 4.1 and 4.3, for $t \in [beg^0, beg^1]$, $|C_i(t) - C_j(t)| < \delta + \epsilon(1 + 4\rho) = \Delta$. Suppose for $0 \le h < f$, and $t \in [beg^h, beg^{h+1}]$, $|C_i(t) - C_j(t)| \le (2km + 1)\Delta + m\delta + 2\rho\epsilon$. We need to prove that for $t \in [beg^f, beg^{f+1}]$, $|C_i(t) - C_j(t)| \le (2km + 1)\Delta + m\delta + 2\rho\epsilon$. We prove it by contradiction.

We assume that for $t \in [beg^f, beg^{f+1}]$, $|C_i(t) - C_j(t)| > (2km + 1)\Delta + m\delta + 2\rho\epsilon$. By Theorem 4.2.2, $\delta + \epsilon(1 + 6\rho)$ is the maximum clock difference if all the synchronizers are nonfaulty. By Lemma 4.2.3, one malicious synchronizer can increase the maximum clock difference by at most $2k\Delta + \delta$, so the maximum clock difference that is greater than $m(2k\Delta + \delta) + \delta + (1 + 6\rho)\epsilon$ can only be accumulated by at least $m + 1$ malicious nodes. However, since there exists at most m malicious nodes, at least one malicious node has served as the synchronizer twice, and increase the maximum clock difference by more than $2k\Delta + \delta$. Suppose it served as r_1-th and r_2-th synchronizer, where $r_2 = r_1 + n \le f$. According to our hypothesis, for $t \in [beg^{r_1}, beg^{r_1+1}]$, $|C_i(t) - C_j(t)| \le (2km + 1)\Delta + m\delta + 2\rho\epsilon$. Within $[beg^{r_1}, beg^{r_2}]$, all the $n - m$ nonfaulty nodes have served as the synchronizer at least once. By Lemma 4.2.5, one nonfaulty node can reduce the maximum clock difference by at least $(k - 1)\Delta$ if the maximum clock difference is greater than $k\Delta$. Because $k = \frac{n - m\frac{\epsilon}{(\delta+\epsilon)}}{n-3m}$, we have $(n - m)(k - 1)\Delta = 2km\Delta + m\delta$, which means $n - m$ nonfaulty nodes can eliminate the clock difference accumulated by m malicious nodes. Thus, one malicious node can contribute at most $2k\Delta + \delta$ into the maximum clock difference, contradicting to the assumption. Thus, we have proved that for $t \in [beg^f, beg^{f+1}]$, $|C_i(t) - C_j(t)| \le (2km + 1)\Delta + m\delta + 2\rho\epsilon$.

Based on Lemma 4.2.6 and Algorithm 4.1, we can see the thresholds on the maximum clock difference and the maximum allowable adjustment are based on the following parameters: δ, ϵ, ρ, n, and m. All the parameters except for δ are either system parameters or measured from the physical characteristics of well-behaved clocks, and thus are bounded. If δ is also bounded, both thresholds will be bounded. As a result, Algorithm 4.1 is a fault-tolerant clock synchronization algorithm when $n > 3m$. Next we show this is indeed the case.

Lemma 4.2.7 *The synchronization interval is bounded. That is, for all $f \ge 1$, $beg^{f+1} - beg^f < y$, and $end^{f+1} - end^f < y$, where $y = ((4km + 2)\Delta + 2m\delta + R)(1 + \rho)$.*

Proof: A nonfaulty node may start its f-th round no earlier than $f \times R - ((2km + 1)\Delta + m\delta)$, and no later than $f \times R + (2km + 1)\Delta + m\delta$ even if it receives no synchronization message.

Suppose node i is the first one to start its f-th clock, we have $C_i^f(beg^f) > f \times R - ((2km+1)\Delta+m\delta)$. If node i is also the first one to start its $(f+1)$-th clock, we have $C_i^f(beg^{f+1}) <$

$(f+1)\times R+(2km+1)\Delta+m\delta$. Then, we have $C_i^f(beg^{f+1})-C_i^f(beg^f) < (4km+2)\Delta+2m\delta+R$. By inequality 2.1, $beg^{f+1} - beg^f < ((4km+2)\Delta+2m\delta+R)(1+\rho)$. If node j (instead of node i) starts its $(f+1)$-th clock first, suppose node i starts its $(f+1)$-th round at beg_i^{f+1}, where $beg^{f+1} < beg_i^{f+1}$. According to $C_i^f(beg_i^{f+1}) - C_i^f(beg^f) < (4km+2)\Delta+2m\delta+R$, we have $beg_i^{f+1} - beg^f < ((4km+2)\Delta+2m\delta+R)(1+\rho)$. Because $beg^{f+1} < beg_i^{f+1}$, we get $beg^{f+1}-beg^f < ((4km+2)\Delta+2m\delta+R)(1+\rho) = y$. Similarly, we can prove that for all $f \geq 1$, $end^{f+1} - end^f < y$.

Lemma 4.2.8 *For all $f \geq 1$, over the time interval $[beg^f, beg^{f+1}]$, $\delta \leq \frac{2\rho R}{1-4\rho(\frac{2nm}{n-3m}+m+1)}$.*

Proof: By inequality 4.1, over the bounded synchronization interval provided by Lemma 4.2.7, the clock drift is at most $\delta \leq 2\rho((4km+2)\Delta+2m\delta+R)(1+\rho)$, where $\Delta = \delta+\epsilon(1+4\rho)$. By a little algebraic calculation, we get $\delta \leq \frac{2\rho(R+(4km+2)\epsilon)}{1-4\rho(2km+m+1)}$. Because $R \gg \epsilon$, by dropping the higher order term $2\rho(4km+2)\epsilon$ compared to $2\rho R$, we have $\delta \leq \frac{2\rho R}{1-4\rho(2km+m+1)}$. From $k > \frac{n-m\frac{\epsilon}{\epsilon+\delta}}{n-3m}$, when using $k > \frac{n}{n-3m}$, we get $\delta \leq \frac{2\rho R}{1-4\rho(\frac{2nm}{n-3m}+m+1)}$.

Theorem 4.2.9 *When $n > 3m$, Algorithm 4.1 is a fault-tolerant clock synchronization algorithm with $(2km+1)\Delta+m\delta+2\rho\epsilon$ as the upper bound of the clock difference and $k\Delta$ as the upper bound of clock adjustment, where $k = \frac{n-m\frac{\epsilon}{(\delta+\epsilon)}}{n-3m}$ and $\Delta = \delta+\epsilon(1+4\rho)$.*

Proof: Trivial based on Lemmas 4.2.6 and 4.2.8.

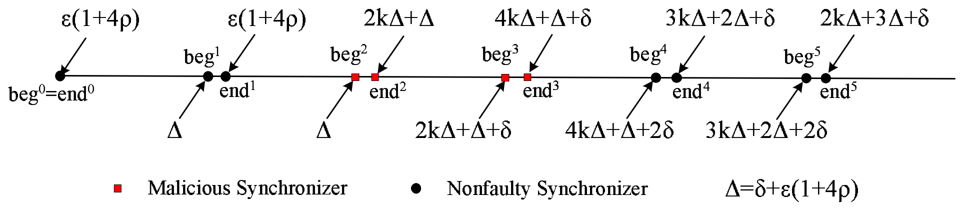


Figure 4.1: Maximum clock difference in cluster-wise time synchronization.

Figure 4.1 shows an example of the changes on the maximum clock difference. The first synchronizer is nonfaulty. During $[beg^1, beg^2]$, the maximum clock difference is less than $\Delta = \delta+\epsilon(1+4\rho)$. The second and the third synchronizers are both malicious, and they collude to increase the maximum clock difference to $4k\Delta+\Delta+2\delta$. The fourth synchronizer is nonfaulty,

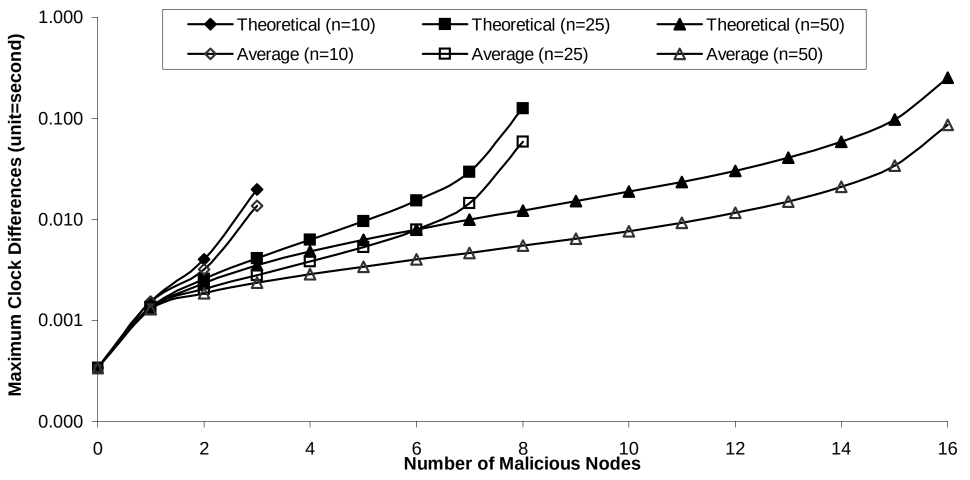


Figure 4.2: Theoretical v.s. average maximum clock differences in simulations.

and decreases the maximum clock difference to at most $3k\Delta + 2\Delta + 2\delta$. We can see that all the malicious nodes can introduce the same amount of maximum clock error into the maximum clock difference, and their order serving as the synchronizer makes no difference.

4.2.4 Discussion

Theoretical v.s. Average Maximum Clock Differences. Theorem 4.2.9 gives an upper bound of the maximum clock difference between nonfaulty nodes when no more than $m < \frac{n}{3}$ nodes are compromised and collude with each other. However, the maximum clock difference is reached only when the m colluding malicious nodes serve as the synchronizer in a row, and the probability that this happens is only $P_m = \frac{(n-m)!m!}{(n-1)!}$.

To understand the maximum clock difference that is generally reached in practice, we performed a series of simulation experiments. Figure 4.2 shows the theoretical maximum clock difference and the average maximum clock difference reached in the simulations. We picked n to be 10, 25, and 50, respectively. For each data point, we used 1,000,000 different random synchronizer orders. The nodes are synchronized once every 2 minutes, the clock drift rate ρ is 10^{-6}, and ϵ is 0.0001 seconds. Our results indicate that when m is greater than 5, the maximum clock difference achieved in the simulations is on average less than half of the theoretical bound.

Combining with MAC Protocols. In a time slotted sensor network, the sensor nodes are divided into clusters, and at any time, only one node in each cluster is allowed to access the wireless communication medium. Time slotted MAC protocols require a local clock synchronization in each cluster to assign time slots to sensor nodes, and our scheme can be used to provide such local clock synchronization. For example, when the time slot size is t_s seconds and each cluster has n nodes, we can set the synchronization interval as $R = k \times n \times t_s$, where k is an integer and $k \geq 1$. Suppose node n_i is assigned the i-th slot for every n time slots. Node n_i can broadcast a synchronization message at $f \times R + i \times t_s$ instead of $f \times R$. It is easy to see that our algorithm can be slightly modified to accommodate this change.

In a CSMA-based sensor network, because all the sensor nodes compete for the wireless communication medium, the assumption that the transmission delay is bounded may not hold. By [31], the transmission delay mainly consists of send time, access time, propagation time, and receive time. Since the send time and receive time can be estimated according to the length of the message, and the propagation time is very small and can be ignored, we only need to deal with the uncertain access time. Thus, we can bound the access time by reserving the wireless channel for the synchronizer to broadcast synchronization messages in a short time interval. It can be achieved by making all the other nodes listen to the channel during the time interval $[f \times R - x, f \times R + x]$, where f is the round number, R is the synchronization interval, and x is the maximum clock difference.

To improve the energy efficiency of sensor networks, several approaches have been proposed to frequently switch sensor nodes into power-saving sleep mode (e.g., [100]). In such approaches, sensor nodes are divided into clusters, and the nodes in the same cluster agree to sleep (or listen) at the same time. When combining our scheme with such power-saving approaches, the only two requirements are (1) that the nodes transmit and listen to others during the live periods and (2) that each round of synchronization can finish during each nonfaulty node's live period. All the nonfaulty nodes satisfy the first requirement in power-saving mode. The second requirement can be satisfied if the maximum clock difference x in our scheme is less than half of the listen time defined in the power-saving approach. Suppose all the nonfaulty nodes are alive during $[f \times R - x, f \times R + x]$. When a nonfaulty synchronizer broadcasts a message at $C(t) = f \times R$, the other nodes are alive since at any time t, $|C_j(t) - C_i(t)| \leq x$ between any two nonfaulty nodes i and j. For example, in [100], the listen time is set to 300 ms, and the sleep time is set to 1 second. According to our simulation result in Figure 4.3, our scheme can guarantee that the maximum clock difference is less than 150 ms. Hence, our scheme can be combined with [100] to provide time synchronization.

Cluster Formation In a large sensor network, it is usually not possible to group all the nodes in the same cluster due to physical constraints such as the communication range. We need to divide the nodes into a number of clusters. Both the number of clusters and the cluster size depend on the node density of the network, the communication range of the sensor nodes, and the requirements of different applications. After the nodes are divided into clusters in which the nodes can communicate with each other through broadcast, our scheme can be used to provide a fault-tolerant cluster-wise time synchronization in each cluster.

In Section 4.3, we propose a secure distributed cluster formation algorithm which can divide a whole sensor network into multiple mutual disjoint cliques even in hostile environments.

4.2.5 Comparison with Previous Techniques

In our proposed algorithm, in each round of time synchronization, only one node serves as the synchronizer, and no other nodes need to respond to the message from the synchronizer. As a result, there will be no collision between the messages involved in time synchronization (when there is no malicious attack). In contrast, almost all of the existing fault-tolerant time synchronization schemes (e.g., CNV [53], HSSD [22]) require the participants send or forward synchronization messages around the same time. Thus, it is very likely to have message collisions in such schemes if they are used in wireless sensor networks.

Moreover, the proposed scheme takes advantage of the broadcast medium as well as a recently proposed authentication technique for sensor networks [104], and thus does not have to use costly digital signatures for broadcast authentication. In comparison, several of the traditional fault-tolerant techniques (e.g., CSM [53], HSSD [22]) require digital signatures, which make them undesirable for resource constrained sensor networks. Note that these schemes cannot use this recent authentication technique [104]. One reason is that they require forwarding of received messages. A malicious node may manipulate a message before forwarding it to other nodes. Another reason is message collision. In a CSMA-based sensor network, all the nodes share the wireless communication channel. In CSM and HSSD, to reduce the synchronization error, after receiving a message, a node will forward the message to other nodes as soon as possible. Therefore, after a node broadcasts a message, since the transmission time is very small, all of its neighbors may receive the message at almost the same time. Suppose all the nodes have the same internal structure, they have a great chance to broadcast messages at the same time, and cause the message collisions.

Table 4.2 compares our scheme with existing fault-tolerant time synchronization algorithms when they are used to synchronize a cluster of n fully connected nodes.

We refer to the maximum number of malicious nodes that one algorithm can tolerate as its *degree of fault-tolerance*. In a cluster of n nodes, our scheme's degree of fault-tolerance is $\frac{n-1}{3}$, which is the same as Algorithms CNV and COM. However, Algorithms CSM and $HSSD$ can

Table 4.2: Performance comparison with traditional fault-tolerant schemes.

Algorithm	Degree of Fault-Tolerance	Comm. Overhead (# msgs/round)	Maximum Clock Difference
CNV [53]	$\frac{n-1}{3}$	n^2 (unicast)	$\frac{n}{n-3m}(2\epsilon + 2\rho R)$
COM [53]	$\frac{n-1}{3}$	n^{m+1} (unicast)	$(6m+4)\epsilon + 2\rho R$
CSM [53]	$\frac{n-1}{2}$	n^{m+1} (unicast)	$(m+6)\epsilon + 2\rho R$
$HSSD$ [22]	$n-1$	n^2 (unicast)	$\epsilon + 2\rho R$
SR	$\frac{n-1}{3}$	1 (broadcast)	$(\frac{2mn}{n-3m}+1)\epsilon + \frac{(\frac{2nm}{n-3m}+m+1)}{1-4\rho(\frac{2nm}{n-3m}+m+1)}2\rho R$

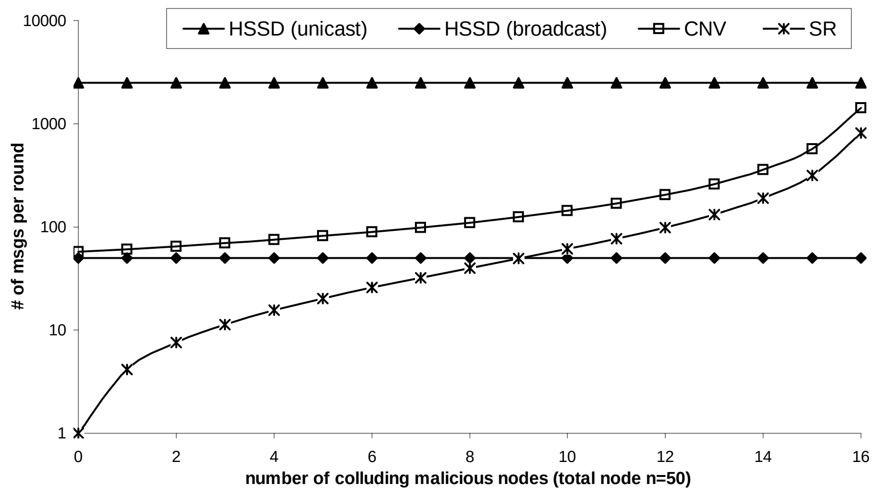


Figure 4.3: Communication overhead with the same guarantee of maximum clock difference

provide better tolerance against colluding attacks.

Now we compare the communication overhead of the proposed scheme with the existing fault-tolerant schemes. To be conservative, we make the assumption that the existing schemes listed in Table 4.2 may use broadcast instead of unicast to send the synchronization messages. This reduces the number of messages per round from n^2 to n for CNV and HSSD, and from n^{m+1} to n^m for COM and CSM. We then set the same bound for the maximum clock difference, and compare the communication overhead in all these schemes. To illustrate clearly the difference, we calculate the communication overhead in these schemes with $n = 50$ and the other parameters the same as those in the simulation experiments (see section 4.2.4).

Figure 4.3 shows the communication overhead in CNV, HSSD, and the proposed scheme for various maximum number of colluding malicious nodes to be tolerated, under the conservative (but unrealistic) assumption that HSSD and CNV can also use authenticated broadcast to send synchronization messages. This figure indicates that the proposed scheme always has less communication overhead than CNV (as well as CSM and COM, which have substantially larger overhead and are omitted from Figure 4.3). Compared with HSSD, the proposed scheme has less communication overhead when the number of colluding malicious nodes to be tolerated is small, but larger communication overhead when the number of colluding nodes grows. However, HSSD has to be modified to reach this performance, because using broadcast in HSSD will cause substantial message collisions. Moreover, the digital signatures required by HSSD make it undesirable for sensor networks, as discussed earlier.

4.3 Secure Distributed Cluster Formation

For a large sensor network, it is not applicable to include all the nodes in one clique, considering the limited wireless transmission range of the sensor nodes. Thus, we need to divide a large network into multiple cliques, in which we can run our fault-tolerant cluster-wise time synchronization scheme. A number of cluster formation protocols have been proposed for wireless sensor networks (e.g., [11, 18, 14, 10, 48, 33, 102, 40, 12, 99, 51, 47, 16]). However, most existing protocols assume benign environments, and are vulnerable to attacks from malicious nodes.

In this section, we propose a secure and distributed cluster formation protocol. By exchanging information with 1-hop neighbors, normal sensor nodes are divided into mutually disjoint cliques, in which all the nodes can directly communicate with each other. Our protocol guarantees that all the normal nodes in each clique agree on the same clique membership even under the attacks from both external and internal malicious nodes. We use the protocol semantics to distinguish malicious behaviors from normal ones, and identify and remove inside attackers that deviate from the protocol.

Our secure cluster formation protocol is different from the authenticated Byzantine Agreement algorithms (e.g., [23, 80, 34]), which can successfully solve the traditional Byzantine General problem [54]. These authenticated Byzantine Agreement algorithms can guarantee all the normal nodes in one group agree on a single or a set of value(s) by using the signature-based authentication. Our protocol aims to divide a sensor network (one large group) into multiple small groups (cliques) and guarantee all the normal nodes in each small group agree on the same group membership. All the normal nodes have to figure out consistently how to partition the network, and the normal nodes in different groups have different group membership.

4.3.1 Problem Statement

Objective: The objective of our clique formation protocol is to divide the normal nodes in a sensor network into mutually disjoint cliques so that all the nodes in the same cliques can directly communicate with each other. Each node should individually compute its *view of clique* based on the information exchanged with its 1-hop neighbors. We denote the view of clique for node i as C_i. For brevity, we call C_i as the *clique* of node i. We call a node a *normal node* if it follows our protocol. Otherwise, it is a *malicious node*. We would like to guarantee that all normal nodes have consistent cliques, as reflected by the following clique agreement property. *Clique agreement* for a normal node i is defined as:

Definition 1 ***(Clique Agreement)*** *For each node $j \in C_i$, $C_j = C_i$.*

Definition 1 implies that for each normal node $j \notin C_i,\ i \notin C_j$ must hold. That is, each normal node belongs to only one clique. Clique agreement is broken if *Clique Inconsistency* is detected. For node i, clique inconsistency is defined as:

Definition 2 ***(Clique Inconsistency)*** *There exists a node $j \in C_i$ such that $C_j \neq C_i$.*

It is desirable that each node can find as large a clique as possible. We do not consider trivial solutions with which each node forms a clique that only includes itself.

Threat Model: We assume an adversary may launch arbitrary attacks against the cluster formation protocol except for completely jamming the communication channel. An external attacker may eavesdrop, inject, and replay packets to disrupt the cluster formation protocol. However, these attacks can be easily defeated with message authentication.

An attacker may generate more severe impact by participating in the clustering formation process using malicious nodes (e.g., those compromised by the adversary). The malicious nodes may arbitrarily deviate from the protocol in order to introduce clique inconsistency. In particular, a malicious node may use directional antenna to send different messages to different neighbor nodes. Moreover, it can communicate with some normal nodes while intentionally keep silence to others. (We call this *silence attack*.) The malicious nodes may launch Sybil attacks [24] or Wormhole attacks [43]. However, we assume these two kinds of attacks can be detected by using the techniques proposed in [73] and [44], respectively.

Assumptions: We assume each node knows its 1-hop neighbors. A message sent by a normal node can be received correctly by all its (1-hop) neighbors in a finite amount of time. We assume each sensor node has a unique ID, and each node can be uniquely identified due to its keying materials (e.g., unique pair-wise keys shared with other nodes, private keys used for digital signatures). All unicast messages exchanged between nodes are authenticated with the key shared between the two nodes.

We assume the sensor nodes can perform public key based digital signature operations. It has been shown in recent investigations [61, 36] that low-end sensor nodes (e.g., MICA2 motes with 8-bit processors) can perform public key cryptographic operations. Moreover, recent development of sensor platforms such as Intel motes[2] uses more advanced hardware, and can perform public key cryptographic operations efficiently.

We use a combination of μTESLA [76] and digital signature to authenticate broadcast messages. We use digital signatures when non-repudiation is necessary, and μTESLA for efficient broadcast authentication in other cases. We assume the clocks of the normal nodes are loosely synchronized, as required by μTESLA. We also assume the public keys used by the sensor nodes are properly authenticated. One approach to ensure this is to issue to each node a certificate for its public key so that other nodes can validate the node's public key by verifying the certificate.

In this section, we first present the details of our protocol, and then analyze its properties in normal situation and hostile environments, including clique consistency property and performance overheads.

4.3.2 The Secure Distributed Cluster Formation Algorithm

Our secure distributed cluster formation protocol consists of five steps. When all the nodes are normal, the cluster formation process terminates after the first four steps. In hostile environments, when clique inconsistency is detected, the protocol provides an extra Step 5 to remove the identified malicious nodes from the network and restart the protocol from Step 1.

The protocol is summarized below:

- **Step 1:** Each node exchanges its neighbor lists with its neighbors, and computes its *local maximum clique*.

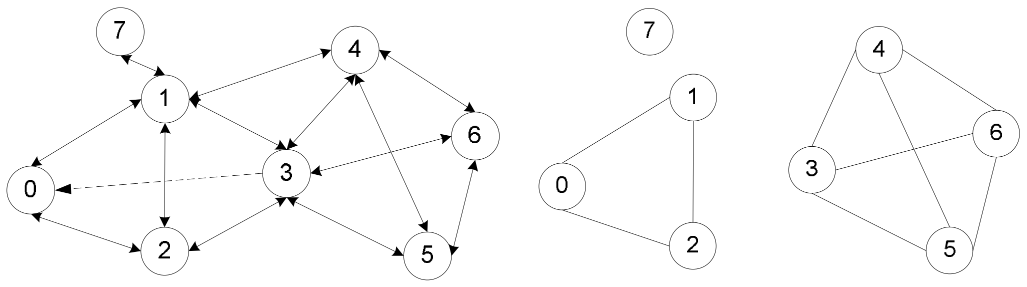
- **Step 2:** Each node exchanges its local maximum clique with its neighbors, and updates its

[2] `http://www.intel.com/research/exploratory/motes.htm`

maximum clique according to its neighbor nodes' local maximum cliques.

- **Step 3:** Each node exchanges the updated clique with its neighbors, and derives its final clique.
- **Step 4:** Each node exchanges the final clique with its neighbors. If no clique inconsistency is detected, it terminates successfully. Otherwise, it enters Step 5.
- **Step 5:** Each node performs conformity checking. If it identifies malicious (neighbor) nodes, it removes them from the network, and restarts the protocol from Step 1. Otherwise, it enforces the clique agreement and terminates.

In the following, we will explain these steps in detail. To facilitate the discussion, we will use the simple example shown in Figure 4.4. Figure 4.4(a) shows a sensor network consisting of 8 sensor nodes. A directional edge from node i to node j represents node j can receive messages from node i. Considering asymmetric communication, we assume node 0 can hear from node 3, while node 3 cannot hear from node 0. Figure 4.4(b) shows the results of our clique formation protocol when all the 8 nodes are normal.



(a) A network with 8 nodes (b) Cluster formation

Figure 4.4: An example of cluster formation

Step 1: Calculating Local Maximum Clique

Based on our assumptions, each node i can obtain a neighbor list L_i that contains the IDs of its 1-hop neighbor nodes. In the first step, all the nodes exchange their neighbor lists with all their neighbors. As discussed earlier, such messages should be authenticated with the pair-wise key shared between neighbors.

After receiving its neighbors' neighbor lists, each node i can build a *neighbor matrix* M_i that records the connectivity between its neighbor nodes. Each element in a neighbor matrix is either 1 or 0. The element in the ith row and jth column of the neighbor matrix is 1 if node i contains node j in its neighbor list, or 0 otherwise. If node i fails to receive the neighbor list from a (previous) neighbor node j, it removes j from its neighbor list.

Each node then symmetrizes its neighbor matrix by considering unidirectional links as no links at all. For example, in Figure 4.4, node 1 considers that node 0 and node 3 are not connected, since node 0 is not in node 3's neighbor list. The neighbor matrix of node 1 in Figure 4.4(a) is shown in Table 4.3.

Table 4.3: Node 1's neighbor matrix in cluster formation process

	0	1	2	3	4	7
0	1	1	1	$1 \Rightarrow 0$	0	0
1	1	1	1	1	1	1
2	1	1	1	1	0	0
3	0	1	1	1	1	0
4	0	1	0	1	1	0
7	0	1	0	0	0	0

Based on the neighbor matrix, each node i individually computes a local maximum clique that includes itself. Based on node i's neighbor matrix, we can construct a graph $G_i = \{V_i, E_i\}$, where V_i consists of node i and its neighbors, and E_i consists of the bidirectional edges between nodes in V_i. It is well known that finding the maximum clique in a random graph is an NP-complete problem [32]. For node i, it is also NP-complete [101] to find the maximum clique containing node i in G_i. To reduce the computation complexity, we propose a heuristic algorithm for node i to compute its local maximum clique, as shown in Algorithm 4.2.

The heuristic algorithm runs in rounds. L_i includes node i's 1-hop neighbor nodes that are eligible to be in the same clique as node i. In each round, node i chooses one neighbor node and adds it into its local maximum clique C_i. Node i maintains a set S_i containing its neighbor nodes that are eligible to be chosen in the next round. Initially, all the neighbors of node i are included in S_i, and C_i only contains node i itself. In the first round, node i computes the number of common neighbors between itself and each neighbor, and finds a neighbor k with the maximum common neighbors $|L_i \cap L_k|$. We use node ID to break the tie. Then node i removes node k from S_i and adds

Algorithm 4.2 Heuristic algorithm to find the local maximum clique

```
INPUT: $G_i = \{V_i, E_i\}, i \in V_i$
OUTPUT: $C_i$
STEPS:
$S_i = \{j|(i,j) \in E_i\}; C_i = \{i\};$
while ( $S_i \neq \emptyset$) do
  Find $k \in S_i$ with maximum $|L_i \cap L_k|$
  $L_i \leftarrow L_i \cap L_k$
  $C_i \leftarrow C_i \cup \{k\}$
  $S_i \leftarrow S_i - \{k\} - \{j|(j,k) \notin E_i, j \in S_i\}$
end while
```

it into C_i. Node i also removes the nodes that are not directly connected with k from set S_i. In the second round, from the updated S_i, node i finds the neighbor node that has the maximum number of common neighbors with all the nodes in C_i (i.e., nodes i and k). Node i then removes this node from S_i and adds it into C_i. Those nodes that are not directly connected with this node will then be removed from set S_i. Node i continues doing so until the set S_i is empty.

After this algorithm finishes, node i sorts the nodes in C_i ascendingly by node IDs and gets its local maximum clique C_i^1. In our protocol, we use C_i^k to denote the clique derived by node i in the kth step ($1 \leq k \leq 4$). Our heuristic algorithm cannot guarantee to find the optimal clique; however, it provides a sub-optimal solution with less computation overhead. We show it through the simulation result in Section 4.3.5

Let us see how this algorithm works on node 1 in Figure 4.4. Initially, node 1 has $C_1 = \{1\}$, $L_1 = \{0, 2, 3, 4, 7\}$, and $S_1 = \{0, 2, 3, 4, 7\}$. In the first round, node 2 has 2 common neighbors $L_1 \cap L_2 = \{0, 3\}$ with node 1; node 3 also has 2 common neighbors $L_1 \cap L_3 = \{2, 4\}$ with node 1. Because node 2 and node 3 have the same maximum number of common neighbors with node 1, we prefer the smaller ID to break the tie. Thus, node 1 adds node 2 into C_1, and $C_1 = \{1, 2\}$. Then, node 1 removes node 2 from S_1, i.e., $S_1 = \{0, 3, 4, 7\}$. Because nodes 4 and 7 cannot directly communicate with node 2, node 1 also removes nodes 4 and 7 from S_1 and $S_1 = \{0, 3\}$. In the second round, node 0 and node 3 have the same number of common neighbors with both node 1 and node 2. Node 1 chooses node 0 that has a smaller ID into C_1. Then, $C_1^1 = \{0, 1, 2\}$, and $S_1 = \emptyset$ after removing node 0 and node 3. Node 3 is removed from S_1 since node 3 is not connected with node 0. Finally, node 1's local maximum clique is $C_1^1 = \{0, 1, 2\}$. Similarly, we have $C_0^1 = C_2^1 = \{0, 1, 2\}$, $C_3^1 = C_4^1 = C_5^1 = C_6^1 = \{3, 4, 5, 6\}$, and $C_7^1 = \{1, 7\}$.

Step 2: Ordering and Updating Maximum Cliques

The local maximum clique computed in step 1 at different nodes are likely to be different. In step 2, each node looks at the local maximum cliques derived by its neighbors, and updates its local maximum clique to prepare for final clique agreement.

In this step, each node i broadcasts its local maximum clique C_i^1 to all its neighbors. For efficiency, such broadcast messages can be authenticated with μTESLA. Because node i calculates its local maximum clique C_i^1 by a heuristic algorithm based on its local neighbor information, it is possible for node i to receive a larger local maximum clique C_j^1 that contains i from a neighbor j. Therefore, after receiving the local maximum cliques from its neighbors, node i checks if there exists any clique C_j^1 which is "better" than its clique C_i^1. To compare cliques computed by different nodes, we define a relation "$\overset{i}{\prec}$" on cliques as follows:

Definition 3 *$C_j \overset{i}{\prec} C_k$ if and only if*

1. *$i \in C_j$, $i \in C_k$, and*

2. *a). $|C_j| < |C_k|$, or*

   b). $|C_j| = |C_k|$, but $c_j < c_k$, where $c_j = min\{a_i | a_i \in C_j \wedge a_i \notin C_k\}$ and $c_k = min\{b_i | b_i \in C_k \wedge b_i \notin C_j\}$, or

   c). $C_j = C_k$, but $j < k$.

The relation $\overset{i}{\prec}$ gives a total order for the local maximum cliques received by node i. We can compare two cliques C_j and C_k by relation $\overset{i}{\prec}$ only if both cliques contain node i. We have $C_j \overset{i}{\prec} C_k$ if the number of nodes in C_k is greater than that in C_j; or both cliques contain the same number of nodes, but for the first two different IDs $c_j \in C_j$ and $c_k \in C_k$ we have $c_j < c_k$; or C_j contains the same nodes as C_k, but $j < k$. In two ascendingly ordered local maximum cliques, the first two different IDs are also the smallest two different IDs. For example, if $C_j = \{1, 2, 3\}$ and $C_k = \{1, 3, 4\}$, then $c_j = 2$ and $c_k = 3$, and $C_j \overset{1}{\prec} C_k$.

Suppose node i receives n cliques that contain node i. Node i orders these cliques as $C^1_{\alpha_1} \overset{i}{\prec} \ldots \overset{i}{\prec} C^1_i \overset{i}{\prec} \ldots \overset{i}{\prec} C^1_{\alpha_n}$, and updates its clique to the "best" clique $C^1_{\alpha_n}$. After Step 2, node i has an updated clique $C^2_i = C^1_{\alpha_n}$. We call C^2_i as node i's *updated clique*.

Let us illustrate this step with the example in Figure 4.4. After receiving the local maximum cliques from neighbor nodes, node 1 has $C^1_0 = C^1_1 = C^1_2 = \{0, 1, 2\}$, $C^1_3 = C^1_4 = \{3, 4, 5, 6\}$, and $C^1_7 = \{1, 7\}$. Node 1 can immediately drop the cliques from nodes 3 and 4, since they do not contain node 1. Because $|C^1_7| < |C^1_0|$, node 1 has $C^1_7 \overset{1}{\prec} C^1_0$. Because $C^1_0 = C^1_1 = C^1_2$ but node IDs $0 < 1 < 2$, we have $C^1_0 \overset{1}{\prec} C^1_1 \overset{1}{\prec} C^1_2$. Therefore, node 1 orders the cliques from node 0, 1, 2 and 7 as $C^1_7 \overset{1}{\prec} C^1_0 \overset{1}{\prec} C^1_1 \overset{1}{\prec} C^1_2$, and updates its clique to $C^2_1 = C^1_2 = \{0, 1, 2\}$. Consider node 7. It will keep its clique unchanged since node 1's clique $C^1_1 = \{0, 1, 2\}$ does not contain node 7. After Step 2, we have $C^2_0 = C^2_1 = C^2_2 = \{0, 1, 2\}$, $C^2_3 = C^2_4 = C^2_5 = C^2_6 = \{3, 4, 5, 6\}$, and $C^2_7 = \{1, 7\}$. We can see that node 7 still has clique inconsistency with node 1.

Step 3: Obtaining Final Clique

In this step, each node i broadcasts its updated clique C^2_i to its neighbors. Similarly to the broadcast messages in step 2, these messages should also be authenticated with μTESLA. For every node j in C^2_i, node i checks if it is included in j's clique C^2_j. If not, node i removes j from its clique C^2_i. After this step, each node i obtains its final clique C^3_i. If node i does not receive node j's updated clique, node i simply keeps node j in its clique.

For our example in Figure 4.4, because $C^2_1 = \{0, 1, 2\}$ does not contain node 7, node 7 removes node 1 from $C^2_7 = \{1, 7\}$, and obtain its final clique $C^3_7 = \{7\}$. Finally, all the nodes are grouped into 3 cliques, which are $C^3_0 = C^3_1 = C^3_2 = \{0, 1, 2\}$, $C^3_3 = C^3_4 = C^3_5 = C^3_6 = \{3, 4, 5, 6\}$ and $C^3_7 = \{7\}$.

If all the nodes are normal, after the first three steps, we can guarantee the clique agreement. We prove this in Section 4.3.3. However, in hostile environments, since compromised nodes may deviate from the protocol, we need extra steps to detect the potential clique inconsistency and identify the malicious nodes.

Step 4: Checking Clique Agreement

All the nodes broadcast their final cliques to their neighbors. Each node i also calculates a secure hash over all the four messages sent in the first four steps, sign this hash value, and append it into the message that contains the final clique. When a normal node i receives the first copy of a final clique C_j^3 from its neighbor j or forwarded by another neighbor, if $j \in C_i^3$, node i rebroadcasts the clique C_j^3. The goal of this rebroadcast is to prevent silence attacks.

Each node i verifies the clique agreement. That is, node i verifies for all $j \in C_i^3$, whether $C_j^3 = C_i^3$ holds. When clique inconsistency is detected, node i enters Step 5; otherwise, it terminates the clique formation process.

Step 5: Identifying Insider or Enforcing Clique Agreement

This step consists of two stages. In Stage I, node i performs *conformity checking* to identify malicious nodes that send inconsistent messages in the previous four steps. The basic idea is to use the protocol semantics to distinguish malicious behaviors from normal ones. When malicious nodes are identified, node i sends an alert to other nodes, using the malicious nodes' signatures as proofs. After removing the malicious nodes from the network, all the remaining nodes restart the protocol from Step 1 again. The malicious nodes that have been identified will be removed from normal nodes' neighbor list and thus cannot launch further attacks.

A malicious node may send messages to some normal neighbor nodes, but keep silence to others. According our assumptions, the messages sent from normal nodes can be received in a finite amount of time. Thus, a normal node may detect a malicious node if certain messages are not received from the malicious node. However, the normal node does not have any proof to convince other normal nodes who do receive the messages from the malicious node. A normal node cannot distinguish a normal node who really detects a malicious node from a malicious node who forges a false alert on a normal node. In such cases, node i enters Stage II to enforce the clique agreement, and finish the clique formation protocol.

We describe these two stages in detail below.

Stage I: Conformity Checking.

Suppose a normal node i detects a clique inconsistency with node j. Node i requests node j to forward the messages that node j received in the first four steps. Because node j has received node i's authenticated final clique C_i^3 in Step 4, only if $C_i^3 \neq C_j^3$, node j will provide its previously received messages to node i. Node j need sign these messages to prove that these messages are forwarded by node j. For efficient signing, node j may calculate a secure hash over all the messages, and simply sign and send this hash value in one message. After verifying node j's signature, node i performs the following conformity checking for node j.

Conformity Checking 1 *Node j follows the clique formation protocol correctly in the first four steps.*

In the above checking, node i re-computes the first three steps of the cluster formation protocol for node j. If the derived final clique is not the same as what node i received from node j in Step 4, node j is a malicious node. Node i can use node j's signatures as a proof to notify other normal nodes in the network. If node j passes checking 1, node i performs the following checking on all the common neighbors of nodes i and j.

Conformity Checking 2 *For any node $k \in L_i \cap L_j$, k sends the same messages to i and j in every step.*

Because node i has messages directly received from node k and the message from node k received and forwarded by node j, if node k sends different messages to nodes i and j in any step, node i can detect the malicious node k and use the conflicting messages from node k as proofs to convince all the other nodes.

Conformity Checking 1 and 2 guarantee to detect the malicious nodes if clique inconsistency is caused by malicious nodes sending inconsistent messages. It is proved by Theorem 4.3.5 in Section 4.3.3. Node i enters Stage II when no malicious node is identified.

Stage II: Consistency Enforcement

When a malicious node launches silence attacks, a normal node may detect the malicious node if certain messages are not received from the malicious node. However, the normal node does not have any proof to convince other normal nodes who do receive the messages from the malicious node. Moreover, a normal node cannot distinguish a normal node who really detects a malicious node from a malicious node who forges a false alert on a normal node.

In such cases, our protocol can ensure that all the normal nodes achieve clique agreement by performing the following consistency enforcement. Suppose two normal nodes i and j find inconsistency, i.e., $j \in C_i^3$, $i \in C_j^3$ (which is proved in Lemma 4.3.4) and $C_i^3 \neq C_j^3$. Without loss of generality, we assume $k \in C_i^3$ and $k \notin C_j^3$.

Consistency Enforcement 1 *If $k \in C_i^2, k \notin C_j^2$, node i receives C_k^1, and node j does not receive C_k^1, then node i removes j from C_i^3, node j removes i from C_j^3.*

Consistency Enforcement 1 deals with the silence attack in Step 2, when a malicious node k sends its local maximum clique to node i and keep silence to node j. However, simply removing k from C_i^3 is not a good option, because node j may be malicious and lie about the receipt of C_k^1. As a result, a normal node k may become isolated. Thus, the safest way is to split nodes i and j into different cliques.

Consistency Enforcement 2 *If $k \in C_i^2 \cap C_j^2$, node j receives C_k^2 and $j \notin C_k^2$, node i does not receive C_k^2, then node i removes k from C_i^3.*

Consistency Enforcement 2 deals with the silence attack in Step 3, when a malicious node k sends its updated clique to node j, but does not send it to node i. Since node k is the only possible malicious node (among nodes i, j, and k), node i simply removes it from C_i^3.

After performing the above two enforcements, we name the new cliques as C_i^* and C_j^* for i and j, respectively. In Section 4.3.3, we prove that our protocol can guarantee clique agreement through these enforcements.

4.3.3 Security Analysis

Effectiveness in Benign Environments

When all the nodes are normal, our protocol guarantees all the nodes in one clique agree on the same clique membership by following the first three steps.

Lemma 4.3.1 *For two nodes i and j, if $i \in C_j^2$ and $j \in C_i^2$, then $C_i^2 = C_j^2$.*

Proof: In Step 2 of our protocol, after node i receives cliques from all its neighbors, it orders these cliques as $C_{\alpha_1}^1 \overset{i}{\prec} \ldots \overset{i}{\prec} C_i^1 \overset{i}{\prec} \ldots \overset{i}{\prec} C_{\alpha_n}^1$, and updates its clique to the "best" clique $C_i^2 = C_{\alpha_n}^1$. Similarly, node j can have an updated clique $C_j^2 = C_{\beta_n}^1$

From $i \in C_j^2 = C_{\beta_n}^1$, node i can compare $C_{\beta_n}^1$ with $C_{\alpha_n}^1$. Node i has $C_{\beta_n}^1 \overset{i}{\prec} C_{\alpha_n}^1$ since $C_{\alpha_n}^1$ is the best clique among from the cliques from all the neighbors. Because $j \in C_i^2 = C_{\alpha_n}^1$, node i can also derive $C_{\beta_n}^1 \overset{j}{\prec} C_{\alpha_n}^1$. However, from $j \in C_i^2$, node j has $C_{\alpha_n}^1 \overset{j}{\prec} C_{\beta_n}^1$. This can happens only if $\alpha_n = \beta_n$, so we can prove $C_i^2 = C_j^2$.

Lemma 4.3.1 guarantees that if node i and node j contain each other in their updated cliques at the end of Step 2, then their updated cliques must contain the same clique membership.

Lemma 4.3.2 *Consider nodes i, j and k, where $k \in C_i^2 = C_j^2$. If $i \notin C_k^2$, then $j \notin C_k^2$.*

Proof: We prove it by contradiction. Suppose $j \in C_k^2$. Because $i \notin C_k^2$ and $i \in C_i^2 = C_j^2$, we have $C_k^2 \neq C_j^2$. Because $k \in C_i^2 = C_j^2$, by Lemma 4.3.1, we have $C_j^2 = C_k^2$. Since $C_k^2 = C_j^2 = C_i^2$, it contradicts to $i \notin C_k^2$.

From Lemma 4.3.1, when node k is included in both node i and node i's updated cliques at the end of Step 2, if node i is not included in node k's updated clique, node j will not be included either. Based on Lemmas 4.3.1 and 4.3.2, we have the following clique agreement theorem that guarantees all the normal nodes in each clique agree on the same clique membership.

Theorem 4.3.3 *For node i and any node $j \in C_i^3$, if all the nodes are normal, we must have $C_i^3 = C_j^3$.*

Proof: For any node $j \in C_i^3$, $j \in C_i^2$ must hold. We also have $i \in C_j^2$, otherwise j should be removed from C_i^3. By Lemma 4.3.1, we have $C_i^2 = C_j^2$. For any node k that $k \in C_i^2$ but $k \notin C_i^3$, we know $i \notin C_k^2$. Then by Lemma 4.3.2, we have $j \notin C_k^2$. Then k will not appear in C_j^3. It means for every node that is removed from C_i^3, it must also be removed from C_j^3. Therefore, we can prove that $C_i^3 = C_j^3$.

Security Analysis in Hostile Environments

Malicious nodes may employ different methods to compromise clique agreement among normal nodes. Our protocol can prevent external attacks by using (unicast and broadcast) message authentication. Thus, a malicious node cannot use a fake identity in our protocol without grasping the keying materials. In the following, we focus on the insider attacks in which some participating nodes are malicious.

If malicious nodes broadcast the same false messages or keep silence to all the normal neighbors, they cannot introduce clique inconsistency. Malicious nodes may send inconsistent messages in different steps, so that the cliques are not correctly derived. However, since such attacks generate the same impact on all the normal neighbors, they cannot introduce clique inconsistency either. Therefore, clique inconsistency can only result from sending different messages to different normal nodes, or launching silence attacks from malicious nodes.

In Section 4.3.3, we prove that malicious nodes will be detected and identified if clique inconsistency is caused by sending inconsistent messages. In Section 4.3.3, we prove that our protocol can tolerate silence attacks and clique agreement can be enforced by removing the conflicting nodes.

Identifying Malicious Nodes

We first introduce Lemma 4.3.4, and then use it to prove Theorem 4.3.5.

Lemma 4.3.4 *For two normal nodes i and j, if $j \in C_i^3$, then we must have $i \in C_j^3$.*

Proof: We prove it by contradiction. Suppose $i \notin C_j^3$. Since $j \in C_i^3$, we must have $j \in C_i^2$. We consider two cases. If $i \notin C_j^2$, j will send C_j^2 to i, then i should remove j from C_i^3 in Step 3. It is contrary to our condition that $j \in C_i^3$. Otherwise, if $i \in C_j^2$ but $i \notin C_j^3$, it means j has removed i from C_j^2. The only reason is that i's clique C_i^2 does not include j, i.e., $j \notin C_i^2$. It contradicts to $j \in C_i^2$.

Lemma 4.3.4 guarantees that if node j is included in node i's final clique, then node j must include node i in its final clique, even in hostile environments.

Theorem 4.3.5 *If clique inconsistency is caused by malicious nodes sending inconsistent messages to different normal nodes, our protocol can identify the malicious nodes.*

Proof: Suppose a normal node i detects clique inconsistency with node j in Step 4, i.e., $j \in C_i^3$ but $C_i^3 \neq C_j^3$. To detect the malicious nodes, node i asks node j to provide its previously received messages and performs Conformity Checking 1 on j. If j passes this checking, it means j follows the protocol correctly, and the inconsistency must come from other nodes. Otherwise, j is malicious.

Consider the case when j performs normally. By Lemma 4.3.4, if normal node $j \in C_i^3$, we must have $i \in C_j^3$. So any node k that is not a common neighbor of both node i and j cannot appear in either C_i^3 and C_j^3. Therefore the inconsistency must come from common neighbors of nodes i and j. By performing Conformity Checking 2 on all the common neighbors of i and j, we will find the different messages sent to i and j, and identify the malicious nodes.

If node j is malicious, node i can detect the conflicts between the messages received from node j in Step 4 and the messages received from node j in Step 5. Because node j provides signatures on these messages, other nodes cannot impersonate it to send fake messages. Thus, node i can use these messages from node j as proofs to inform other nodes in the network. The malicious node j will be removed from the network. Similarly, if a common neighbor node k of node i and node j is malicious, node i can use the messages directly received from node k and node k's messages received and forwarded by node j as proof to remove node k from the network.

Enforcing Clique Agreement

We observe that silence attacks can introduce clique inconsistency only in Steps 2 and 3. In Step 1, a malicious node may send its neighbor list to some neighbor nodes, but withhold it from other neighbor nodes. However, in Step 2, our protocol allows a normal node i update its clique to a “better” clique, even if the better clique contains some nodes that did not send their neighbor lists to node i in Step 1. Thus, the silence attack in Step 1 will not cause clique inconsistency.

In Step 2, clique inconsistency can only come from the “better” cliques sent by malicious nodes, since a normal node will update its clique to a “better” clique. Suppose nodes i and j are normal. A malicious node k may send i a “better” clique C_k^1 that includes i and j, but withhold the message from node j. Then node i updates its clique to C_k^1. If node j receives the “better” clique from node i, it updates its clique to C_i^1. Therefore, node i and j include each other in their cliques that are inconsistent. However, Consistency Enforcement 1 can remove such clique inconsistency.

In Step 3, clique inconsistency can only be introduced by removing nodes from cliques. Suppose $k \in C_i^2 \cap C_j^2$. In Step 3, node k can send a clique to remove itself from i's clique, while keeping silence to j. Then the final clique of j contains k, which is not in node i's final clique.

In Step 4, after a normal node i receives a final clique C_k^3 from node k, node i rebroadcasts C_k^3 if $k \in C_i^3$. Because we assume the messages from a normal node can be received correctly by its normal neighbors, this rebroadcast can guarantee that if one normal node receives C_k^3 from node k, all the other normal nodes in the same clique can receive C_k^3. Thus, it can prevent silence attacks in Step 4.

In the following Theorem 4.3.6, we prove that by removing the inconsistent nodes from cliques through the consistency enforcement, all the normal nodes can achieve clique agreement even if malicious nodes intentionally keep silence to certain normal nodes.

Theorem 4.3.6 *For any two normal nodes i and j, after Step 5, if $j \in C_i^*$, we have $C_i^* = C_j^*$.*

Proof: We prove it by contradiction. Suppose $C_i^* \neq C_j^*$. Since our protocol can only remove nodes from cliques when inconsistency is detected, C_i^3 must contain all the nodes in C_i^*. Therefore $j \in C_i^3$. By lemma 4.3.4, we have $i \in C_j^3$. We consider two cases.

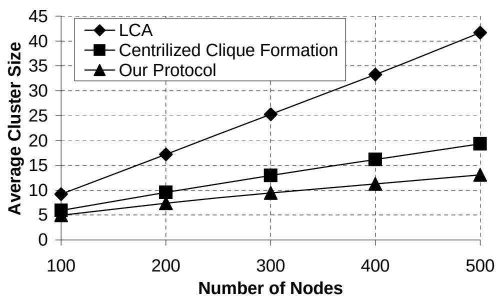
First, suppose $C_i^3 \neq C_j^3$ and $C_i^* \neq C_j^*$. Without loss of generality, we assume node $k \in C_i^3$ but $k \notin C_j^3$. Nodes i and j find inconsistency after exchanging C_i^3 and C_j^3. By Consistency Enforcement 1, node i removes j from its clique, and node j also removes i from its clique. Therefore we have $j \notin C_i^*$. It is contrary to the condition $j \in C_i^*$.

Second, we assume $C_i^3 = C_j^3$, but $C_i^* \neq C_j^*$. Without loss of generality, suppose node $k \in C_i^*$ but $k \notin C_j^*$. Because nodes can only be removed to enforce clique agreement in Step 5, k cannot be added to C_i^*, but removed from C_j^*. This means C_k^3 is inconsistent with C_j^3. Since $C_i^3 = C_j^3$, C_k^3 is also inconsistent with C_i^3. Because node j re-broadcasts the clique C_k^3 received from k, node i will receive C_k^3 even if node k keeps silence to i. Thus, i should remove k from C_i^*. We find contradiction.

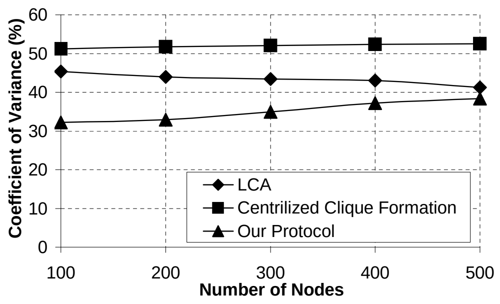
In our protocol, the clique consistency checking is only performed in Step 4, though it can be executed in each step. The reason is to reduce the computation overhead by decreasing the number of signature generation/verification. Each node need not verify the signatures from other nodes unless it detects clique inconsistency. Even if clique inconsistency is detected, each node only generates and verifies the signatures of the messages exchanged in Step 4 and Step 5. If the protocol checks the consistency in every step, malicious nodes may be detected in an earlier step. However, the computation overhead will be increased a lot.

4.3.4 Performance Analysis

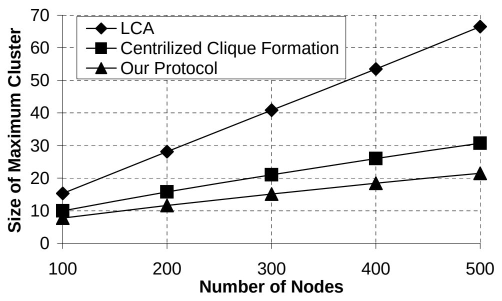
Computation Overhead: We make several efforts to lower the computation overhead in our protocol. In all the five steps, each node i uses μTESLA to authenticate its broadcast messages. Because μTESLA uses secure key cryptography that has much less computation overhead than public key cryptography, we only analyze the computation overhead on public key operations.



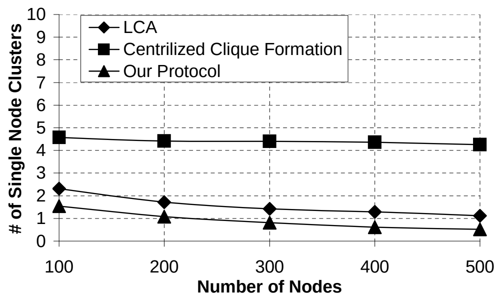
(a) Average Cluster Size



(b) CV (%) on Cluster Size



(c) Size of the Maximum Cluster



(d) # of Single-Node Clusters

Figure 4.5: Comparison of cluster metrics

In Step 4, each node i signs the secure hash of its local messages sent in the first four steps, instead of signing each message individually. Each node need not verify the signatures from other nodes unless it detects clique inconsistency with them. Therefore, in benign environments, no signature verification is necessary. In hostile environments, after detecting a clique inconsistency with node j, node i verifies the signature from node j. In Step 5, after receiving node i's request, node j generates a signature on the secure hash over the previous received messages from its neighbors. Then, node i needs to verify node j's signature on the forwarded messages. If node j passes Conformity Checking 1, node i needs to verify $|L_i \cap L_j|$ signatures from the common neighbors of i and j.

Because a node may verify more messages than those it signs, we propose to choose public key cryptosystems with a fast decryption speed, such as RSA, which can verify one signature in 0.43s on ATmega128 [36]. Since the clique formation process will not be performed frequently, the computation overhead is acceptable for sensor nodes.

Communication Overhead: Each node i broadcasts one message in each of the first three steps. In

Step 4, besides broadcasting its final clique, node i also rebroadcasts the first copy of the final clique message about a neighbor in node i's final clique C_i^3. In total, node i sends $|C_i^3| + 3$ messages.

Suppose node j has $|L_j|$ neighbors. When node i detects a clique inconsistency and requests node j to forward its previously received messages in Step 5, node j needs to forward $4|L_j|$ messages received in the first four steps, plus one message including the signature for the secure hash over all the forwarded messages.

Storage Overhead: According to the analysis of the computation overhead, each node i should store all the $4|L_i|$ messages received in the four steps, where $|L_i|$ is the neighbor number of node i. When node i detects a clique inconsistency with node j, node i needs to store $4|L_j| + 1$ messages from node j. Node i can release the memory after verifying these messages.

4.3.5 Experimental Results

Through simulation, we show that our protocol can provide secure cluster formation without sacrificing the performance of the clusters. We use the following metrics to evaluate the cluster characteristics: *average cluster size*, *maximum size of clusters*, *variance of the cluster size*, and *number of single-node clusters*.

The average cluster size depends on the density of the networks and the transmission range of the sensor nodes. The average cluster size should not be too small. In sensor networks, it is not desirable to include too many nodes in a large cluster due to the increasing message collisions and transmission delay in a large cluster. We use Coefficient of Variance (CV) = 100*(Standard Deviation)/(mean value of set) to evaluate the variance of the cluster size. We expect to divide nodes into clusters with a low coefficient of variance. A cluster formation protocol should minimize the clusters with a single node.

In our simulation, we uniformly deploy 100, 200, 300, 400 and 500 sensor nodes in a 100 $\times$ 100 (m^2) simulation area, respectively. The transmission range of all the sensor nodes is fixed to 20 meters. Each point in the result figures is the average result of 1000 experiments.

We compare the cluster characteristics of our distributed protocol to LCA [11], one typical Leader-First based cluster formation protocol, and a centralized clique formation protocol. In LCA, from the lowest ID node to the highest ID node, a node declares itself to be a cluster-head if it has the lowest ID among the non-covered neighbor nodes. A node is covered if it is in the 1-hop neighborhood of a node who has declared itself to be a cluster-head. In the centralized clique formation protocol, we assume a sink node has obtained the topology graph G of the whole network.

The sink node first finds the maximum clique C_1 in G, and updates G by removing C_1 from G. Then, it finds the maximum clique C_2 in the remaining G, and then removes C_2 from current G. The algorithm completes when G becomes empty. We borrow the C implementation (dfmax) from [2] to find a maximum clique in a random graph.

Figure 4.5 compares the cluster characteristics of three protocols. As Figure 4.5(a) shows, the average cluster sizes of the three protocols increase with the node density of the network. Our protocol has a smaller average cluster size than the other two protocols. The reason is that our protocol requires all the nodes in a clique be able to directly communicate with each other. While, in LCA, the maximum distance between any two nodes in one cluster is two hops. Compared to the centralized clique formation protocol, our heuristic protocol in Step 1 may not find the maximum local clique. Thus, the average cluster number is a little smaller.

Figure 4.5(b) shows the variance of the cluster sizes. Our protocol has a smaller coefficient of variance than the other two protocols, which means our protocol generates more uniform clusters. Figure 4.5(c) presents the maximum cluster sizes in three protocols. Our protocol has a moderate maximum cluster size. As Figure 4.5(d) shows, our protocol has fewer single-node clusters than the other two protocols. The reason is that LCA and the centralized clique formation protocol attempt to form the largest cluster first, and thus leave some nodes into small clusters. While in our protocol, because all the nodes choose their clusters in a distributed and parallel way, it decreases the chances to form large clusters and single-node clusters.

4.4 Summary

In this chapter, we developed a fault-tolerant cluster-wise time synchronization scheme that can guarantee an upper bound of clock difference between any nonfaulty nodes in a cluster, provided that the malicious nodes are no more than one third of the cluster. Compared with the existing fault-tolerant clock synchronization techniques, the proposed scheme can avoid the message collision problem in these techniques, and does not require costly digital signatures. The proposed scheme also has some limitations. First, it requires that the nodes in a cluster maintain initial synchronization. Thus, it has to rely on other means, for example, a bootstrapping phase with trusted external devices (see Section 4.2.1), or a fault-tolerant initial clock synchronization method (e.g., [59]). Second, it requires that each node be able to reach all the other nodes in a cluster, thus reducing the geographical coverage of each cluster.

We also proposed a secure and distributed clique formation protocol for sensor networks to divide sensor nodes into mutually disjoint cliques, in which we can run the fault-tolerant cluster-wise time synchronization to achieve a consistent group clock time. The protocol is fully distributed, and guaranteed to terminate. Currently, our clique formation scheme is only suitable for static sensor networks. It requires to use digital signatures to identify the possible malicious nodes; however, digital signatures are still quite heavy for computation restrained sensor nodes.

Chapter 5

Secure and Resilient Global Time Synchronization with Unicast Authentication

With the secure single-hop pair-wise time synchronization in Chapter 3, a compromised node has limited impact on single-hop time synchronization between neighbor nodes. It can only affect the clock difference between itself and a normal node (rather than between normal nodes). However, when a pair of nodes are synchronized through a multi-hop path (e.g., [26, 31, 87]), a compromised node in the path can introduce an arbitrary error. This implies global time synchronization using multi-hop paths is vulnerable to compromised nodes.

In this chapter, we develop two secure time synchronization schemes, *level-based time synchronization* and *diffusion-based clock synchronization*, to provide secure multi-hop pair-wise and secure global time synchronization. The basic idea of both schemes is to provide redundant ways for one node to synchronize to a far-away node, so that it can tolerate partially missing or false synchronization information provided by the malicious nodes. To achieve global clock synchronization, we adopt a model where all the sensor nodes synchronize their clocks to a common source, which is assumed to be well synchronized to an external clock. The level-based scheme builds a level hierarchy in the sensor network, and then synchronizes the whole network level by level. The diffusion-based scheme allows each node to diffuse its clock to its neighbor nodes after it has synchronized to the source node. Our analysis and simulation results indicate that these two approaches are complementary. The level-based approach is suitable for static sensor networks, while the diffusion-based approach is suitable for dynamic sensor networks. The level-based approach has less overhead and higher precision than the diffusion-based approach, but has less coverage than the diffusion-based approach.

To improve the synchronization precision and reduce communication overhead in large sensor networks, we propose to deploy multiple source nodes in the network, so that the sensor nodes can synchronize to the nearest source node. Moreover, we extend this approach to increase the resilience of such time synchronization. As a result, a sensor node can obtain the correct clock time even if up to the half of the source nodes to which it can synchronize are compromised.

We assume each pair of nodes communicate through unicast for both pair-wise and global time synchronization, and any two nodes that need to communicate with each other share a unique pair-wise key, so that the messages between them are authenticated. One node can also identify the other node based on the unique pair-wise key. Such pair-wise keys can be provided by several key predistribution schemes proposed for sensor networks recently (e.g., [56, 17, 25]).

5.1 Global Time Synchronization Model

5.1.1 A Motivating Example

Consider Figure 5.1, in which there are multiple, interleaved paths between node S and node D. Assume node D needs to estimate the clock difference between itself and node S. Suppose that each pair of nodes connected by an edge in the network are neighbors, and have synchronized with each other using the single-hop pair-wise time synchronization scheme in Chapter 3. For convenience, we denote the pair-wise clock difference between any two nodes i and j as $\delta_{i,j}$. Specifically, $\delta_{i,j} = C_j - C_i$, where C_i and C_j are the local clock of node i and node j, respectively. We assume some nodes may have been compromised, and thus may lie about any information needed by other nodes.

We first estimate the clock differences between S and the nodes close to S (in a fault-tolerant way), then gradually use these clock differences to estimate those between S and the nodes

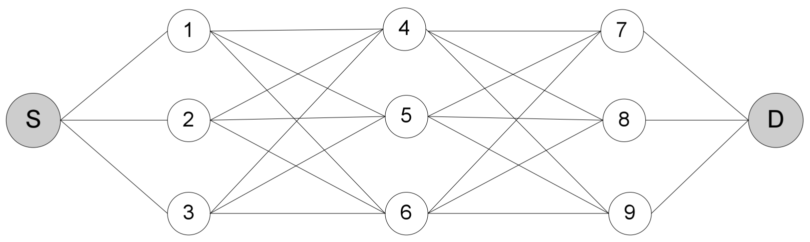


Figure 5.1: A mesh network between nodes S and D

farther away from S, and eventually derive the clock difference between S and D. According to the assumption, nodes 1, 2, and 3 have obtained $\delta_{1,S}$, $\delta_{2,S}$, and $\delta_{3,S}$, respectively. Now consider node 4. Node 4 may estimate $\delta_{4,S}$ through 1, 2, or 3. To deal with potentially malicious nodes, node 4 can estimate $\delta_{4,S}$ through all three nodes. When node 1 is chosen, node 4 can easily compute $\delta_{4,S}^{(1)} = \delta_{4,1} + \delta_{1,S}$. Similarly, node 4 can compute $\delta_{4,S}^{(2)}$ and $\delta_{4,S}^{(3)}$ through nodes 2 and 3, respectively. Then node 4 chooses the median of the three values as $\delta_{4,S}$. As a result, if only one of nodes 1, 2, and 3 is malicious and attempts to attack time synchronization, its effect will be removed.

This process may continue for nodes 7, 8, and 9, assuming 4, 5, and 6 have obtained $\delta_{4,S}$, $\delta_{5,S}$, and $\delta_{6,S}$, respectively. Eventually, node D can obtain the correct clock difference $\delta_{D,S}$ if there is at most one malicious node in each level in the mesh network between S and D. In general, if there are $2t+1$ nodes in each level of the mesh network between nodes S and D and all the neighboring nodes can communicate with each other, this approach can tolerate up to t colluding malicious nodes in each level.

5.1.2 Our Model

We develop our secure time synchronization techniques by generalizing the above motivating example. We assume there is a *source node S* that is well synchronized to the external clock, for example, through a GPS receiver. We would like to synchronize the clocks of all the sensor nodes in the network to that of the source node. We assume the source node is trusted, and all the other nodes know the identity of the source node.

We adopt the following model for secure and resilient global time synchronization:

1. Each node i maintains a *local clock* C_i. The local clock of the source node (i.e., C_S) is the desired global clock.

2. For each neighbor node j, each node i maintains a *single-hop pair-wise clock difference* $\delta_{i,j} = C_j - C_i$ with the secure single-hop pair-wise time synchronization technique in Chapter 3.
3. Each node i also maintains a *source clock difference* $\delta_{i,S}$ between its local clock and the clock of the source node S. Node i can directly obtain it if it is a neighbor node of S. Otherwise, node i needs to estimate $\delta_{i,S}$.
4. To tolerate up to t malicious neighbor nodes, each node i needs to compute at least $2t + 1$ *candidate* source clock differences through different neighbor nodes. Specifically, the candidate source clock difference obtained through neighbor node j is $\delta_{i,S}^{(j)} = \delta_{i,j} + \delta_{j,S}$. Node i then chooses the median of the candidate source clock differences as $\delta_{i,S}$. We assume the sensor network of concern is dense so that each node has enough number of neighbor nodes to obtain $2t + 1$ candidate source clock differences.
5. Each node i can estimate the *global clock* C_S by using its local clock and its source clock difference (i.e., $C_S = C_i + \delta_{i,S}$).

We assume there are malicious nodes (e.g., compromised nodes that possess valid cryptographic keys) in the network, which may collude together to disrupt time synchronization. A malicious node i may affect a normal node j by affecting node j's measurement of $\delta_{i,j}$ and/or lying about $\delta_{i,S}$. Our goal is to provide secure time synchronization so that even if a certain number of malicious nodes collude together to disrupt clock synchronization, each normal node i can still synchronize its local clock to the source node.

We give the following recursive definition to further clarify the correctness of secure and resilient time synchronization.

Definition 4 *With a unique source node S, a source clock difference $\delta_{i,S}$ obtained by node i is* correct *if*

- *node i is a neighbor node of node S, or*
- *$\delta_{i,S}$ is computed as $\delta_{i,S} = \delta_{i,j} + \delta_{j,S}$, where node j is a neighbor of node i, and either (1) node j is a normal node and $\delta_{j,S}$ is correct, or (2) node i has two other normal neighbor nodes m and n such that $\delta_{m,S}$ and $\delta_{n,S}$ are correct and $\delta_{i,m} + \delta_{m,S} \leq \delta_{i,S} \leq \delta_{i,n} + \delta_{n,S}$.*

It is easy to see that if node i has a correct source clock difference, it can estimate the global clock C_S "correctly".

We assume each pair of nodes communicate through unicast for both pair-wise and global time synchronization, and any two nodes that need to communicate with each other share a unique pair-wise key, so that the messages between them are authenticated. One node can also identify the other node based on the unique pair-wise key. Such pair-wise keys can be provided by several key predistribution schemes proposed for sensor networks recently (e.g., [56, 17, 25]). For brevity, we assume all pair-wise clock difference $\delta_{i,j}$ between two neighbor nodes i and j is obtained with our secure single-hop pair-wise time synchronization technique in Chapter 3 without explicit statement. Thus, the single-hop pair-wise clock difference between two normal nodes is always trusted, though it may be impaired when one of the nodes is malicious.

It is natural for sensor nodes to communicate through broadcast, but in hostile environments, it requires local broadcast authentication. Recent research (e.g. TinyPK [97]) shows that it is applicable to apply asymmetric cryptographic technology in sensor network. However, due to the resource constraint of sensor nodes, those techniques are vulnerable to DoS attacks. In our later research, we develop a secure and resilient global time synchronization using broadcast authentication based on a novel use of the μTESLA broadcast authentication protocol for *local authenticated broadcast*, resolving the conflict between the goal of achieving time synchronization with μTESLA-based broadcast authentication and the fact that μTESLA requires loose time synchronization. We will discuss the detail in Chapter 6.

5.2 Secure and Resilient Global Time Synchronization

We develop two secure and resilient time synchronization schemes for sensor networks: the level-based scheme and the diffusion-based scheme. In the level-based scheme, a level hierarchy is established in the sensor network, and each node obtains the clock differences from its parent nodes in the level hierarchy. In the diffusion-based scheme, a node can obtain the clock differences from any neighbor nodes. The level-based scheme is suitable for static sensor networks, where sensor nodes stay in the same places after deployment; whereas the diffusion-based scheme is more suitable for dynamic sensor networks, where sensor nodes may move frequently.

5.2.1 Level-Based Time Synchronization

Level-based time synchronization aims at static sensor networks, where the network topology does not change frequently. Level-based time synchronization consists of two phases: *level discovery phase* and *synchronization phase*. The level discovery phase is to organize the legitimate sensor nodes into a hierarchy rooted at the source node S so that two nodes connected in the hierarchy are neighbors. Each node except for the root has a set of parent nodes in the hierarchy, and each node except for the leaf nodes has a set of children nodes. Each node is also associated with a *level*, which is the number of hops in the longest path from the root to this node. We refer to this hierarchy as the level hierarchy. In the synchronization phase, all the sensor nodes obtain the source clock differences through their parent nodes, estimate their own source clock differences, and then help their children nodes to synchronize their clocks.

Level Discovery Phase

To establish the level hierarchy, each node maintains three variables: `level`, `parents`, and `children`. The variable `level` records the level of the node. `Parents` and `children` record the parents and the children of the node in the level hierarchy, respectively. After the level hierarchy is established, a node i can obtain the candidate source clock differences from the nodes in its parent set, and may help the nodes recorded in its children set to obtain their source clock differences.

We assume all the sensor nodes have discovered their neighbors before the level discovery phase. Consider the source node S. Initially, S.`level` $= 0$, S.`parents` $= \emptyset$, and S.`children` $= \{x | x$ is a neighbor of $S\}$. The variables of all the other nodes are unknown. The source node S initiates the level discovery phase by unicasting a *level discovery message* to each of its neighbor nodes. A level discovery message contains the sender's identity and its level number, authenticated (and optionally encrypted) with the pair-wise key shared between the sender and the receiver. After receiving an authenticated level discovery message from S, each neighbor i of S sets i.`level` as 1, and i.`parents` as $\{S\}$. It then unicasts a level discovery message to each of its neighbor nodes except for S.

The nodes that are more than one hop away from the source node may receive more than one level discovery messages from their neighbor nodes. To tolerate up to t malicious parent nodes in the synchronization phase, a node needs to record $3t + 1$ parent nodes that will send synchronization message to it. When a normal node has $3t + 1$ parent nodes in the level hierarchy, even if up to t malicious parent nodes keep silent during the synchronization phase, the node still can receive $2t + 1$ candidate source clock differences and synchronize its clock. We have two options for a sensor node to obtain its level and parent set. In the first option, after receiving authenticated level discovery messages from the first $3t + 1$ different neighbor nodes, node i chooses these nodes as its parent nodes. In the second option, node i may wait for a period of τ time units after getting the first $3t + 1$ candidate parent nodes, and then choose the $3t + 1$ nodes with the least levels as the parent nodes. When using the second option, the convergence time of the level discovery phase is longer than that by using the first option, but the average level of the sensor nodes is smaller than that by using the first option. Because the source node runs level discovery process infrequently, we adopt the second option in our level-based scheme. Assuming the maximum level of the parent nodes is l, node i then sets $i.\texttt{level}$ as $l + 1$.

After determining its level, a node i unicasts level discovery messages to its neighbor nodes from which it has not received any authenticated level discovery message. Node i also unicasts messages to its parent nodes to add itself as one of their children nodes. Node i will drop subsequent level discovery messages.

The level hierarchy needs to be maintained when there are slight changes in the network (e.g., node joins, failures). The maintenance may be performed locally without re-executing the level discovery phase. When a new node joins the network, it needs to determine its level and find its parent nodes in the level hierarchy. To do it, it unicasts *level query messages* to all its neighbor nodes. A neighbor node will send back a *level reply message*, containing its identity and its level. All the messages are authenticated by the shared pair-wise key. The new node can determine its level and parent nodes by the receiving level reply messages. In the synchronization phase, when a node fails to receive from at least $2t + 1$ parent nodes in several rounds of synchronization, it will send level query messages to its neighbor nodes that are not its parent or children nodes, and recruits new parent nodes according to the level reply messages.

Synchronization Phase

Due to the clock drift of sensor nodes, the source node S periodically initiates the synchronization phase by unicasting *synchronization messages* to its neighbor nodes. A synchronization message contains the sender's identity, a sequence number, and the sender's source clock difference. Each node maintains a sequence number, and increases it in each round of synchronization.

These nodes then further send synchronization messages to their children nodes. All the relevant messages are authenticated with a key shared between the communicating nodes.

After receiving a synchronization message from node S, level one nodes start the single-hop pair-wise time synchronization with the source node. Then, they unicast synchronization messages to their children nodes. Consider a node i at a level greater than 1. When it receives a synchronization message from a parent node j, after obtaining the single-hop pair-wise clock difference from node j, node i calculates a candidate source clock difference by $\delta_{i,S}^{(j)} = \delta_{i,j} + \delta_{j,S}$. To tolerate up to t malicious nodes in its parent nodes, it has to collect at least $2t + 1$ candidate source clock differences through its parent nodes. Node i sets the source clock difference $\delta_{i,S}$ as the median of the $2t + 1$ candidate source clock differences. Then, node i unicasts its source clock difference to its children nodes.

Effectiveness

We first introduce Lemma 5.2.1 to facilitate the analysis.

Lemma 5.2.1 *Assume a normal node i has at least $2t+1$ neighbor nodes, among which there are at most t colluding malicious nodes. Node i can obtain a correct source clock difference if it receives from each neighbor node the source clock difference and all the normal neighbor nodes provide their correct source clock differences.*

Proof: According to our model, node i computes a candidate source clock difference with the source clock difference provided by each neighbor node, and then chooses the median as its source clock difference $\delta_{i,S}$. Suppose the source clock difference is obtained through node j, that is, $\delta_{i,S} = \delta_{i,j} + \delta_{j,S}$. There are two cases. (1) If node j is a normal node, both $\delta_{j,S}$ and $\delta_{i,j}$ must be correct according to the assumption, and $\delta_{i,S} = \delta_{i,j} + \delta_{j,S}$ is correct according to Definition 4. (2) Suppose node j is malicious. Because there are at most t malicious nodes, $\delta_{i,S}$, which is the median of the $2t + 1$ candidate source clock differences, must be between two candidate source clock differences obtained through two normal nodes. Thus, the source clock difference $\delta_{i,S}$ is still correct, according to Definition 4.

Based on Lemma 5.2.1, we have the following results on the effectiveness of level-based time synchronization.

Lemma 5.2.2 *The level-based time synchronization can synchronize all the normal nodes correctly, if each normal node at level l ($l > 1$) receives at least $2t + 1$ source clock differences from distinct parent nodes and at most t out of these parent nodes are colluding malicious nodes.*

Proof: This is equivalent to proving that each normal node i can obtain the correct source clock difference $\delta_{i,S}$ if the given conditions are satisfied. We prove it by induction.

Each node i at level one can obtain the correct source clock difference $\delta_{i,S}$, which is the single-hop pair-wise clock difference. Now suppose each normal node at a level less than or equal to level k ($k \geq 1$) has obtained the correct source clock difference. Consider a normal node j at level $k + 1$. All parents of node j have levels less than or equal to k. If node j receives source clock differences from at least $2t + 1$ distinct parent nodes and at most t out of them are colluding malicious nodes, then by Lemma 5.2.1, node j can obtain its correct source clock difference $\delta_{j,S}$.

5.2.2 Diffusion-Based Time Synchronization

With level-based time synchronization, all the sensor nodes synchronize to the source node by using the level hierarchy. The following diffusion-based time synchronization scheme allows sensor nodes to obtain source clock differences through any neighbor nodes without requiring any level hierarchy.

In the diffusion-based scheme, the source node S initiates the synchronization process periodically by unicasting synchronization messages to its neighbor nodes. After obtaining a source clock difference from the source node, the neighbor nodes of S update their source clock differences, and then unicast synchronization messages to their neighbors except for S. To tolerate up to t colluding malicious nodes among its neighbor node, a node more than one hop away from the source node needs to receive at least $2t + 1$ candidate source clock differences through different neighbor nodes, and updates its source clock difference as the median of the $2t + 1$ source clock differences. The node then sends synchronization messages to its neighbors from which it has not received synchronization messages.

We have the following results on the effectiveness of diffusion-based time synchronization.

Lemma 5.2.3 *The diffusion-based time synchronization scheme can synchronize all the normal* *nodes correctly, if each normal node that is more than one hop away from the source node receives the source clock differences (of the neighbor nodes) from at least $2t + 1$ distinct neighbor nodes among which at most t nodes are colluding malicious nodes.*

Proof: This is equivalent to proving that each node i can obtain the correct source clock difference $\delta_{i,S}$ if the given conditions are satisfied. We prove it by induction.

Each neighbor node i of the source node can obtain the correct source clock difference $\delta_{i,S}$, which is the single-hop pair-wise clock difference. Thus, all normal nodes have correct source clock differences right after S's neighbor nodes obtain their source clock differences. Assume at a certain time, all the normal nodes that have been synchronized have correct source clock differences. Consider a normal node j that is more than one hop away from the source node. From the assumption, if it can receive the source clock differences (of the neighbor nodes) from at least $2t + 1$ distinct neighbor nodes, among which at most t nodes are colluding malicious nodes, then by Lemma 5.2.1, node j can obtain its own correct source clock difference.

The benefit of the diffusion-based scheme is that all communication is localized without depending on a distributed level hierarchy. However, a node has to send synchronization messages to all its neighbor nodes from which it has not received synchronization messages. The diffusion-based scheme potentially has higher communication overhead than the level-based ones, but it is more applicable for dynamic sensor networks, where the network topology changes frequently.

5.2.3 Security Analysis

By using unique shared pair-wise keys for message authentication, our scheme can prevent external malicious nodes from inserting or modifying messages and impersonating other nodes, and internal malicious nodes from pretending to be other nodes. Next we analyze other possible attacks against the proposed schemes, and show how our schemes can prevent or tolerate these attacks.

Attacks against Level Numbers This attack is unique to the level-based scheme. During level discovery, a malicious node may lie about its level to normal nodes. Because a node sets its level as the maximum level of the parent nodes plus one, when a malicious node sends a level discovery message with a large fake level, the normal node will assign itself a large level if it chooses the malicious node as one of its parent nodes. This problem can be mitigated if the normal node waits for a period of time, since there may be other normal candidate parent nodes at lower levels. However, there is in general no guarantee that this will happen. Alternatively, we can set a level threshold to the maximum level in the level hierarchy. When a node receives from a parent node whose level is greater than the level threshold, it drops the message.

Silence Attacks A malicious node may delay or refuse to provide source clock differences to its children nodes in the synchronization phase. In the level-based scheme, we can tolerate such attack by record $3t + 1$ parent nodes in the parent set, so that even if up to t malicious nodes keep silence, a normal node can still receive $2t + 1$ source clock differences. This attack has little effect on diffusion-based scheme when a normal node can obtain source clock differences from any $2t + 1$ neighbor nodes, though the malicious nodes keep silience.

Replay Attacks A malicious node may launch replay attacks during the synchronization process. Specifically, a malicious node may record a synchronization message in one round of time synchronization, and replay it to normal nodes in later rounds. As a result, the normal nodes may accept the replayed message, and derive a false source clock difference.

This attack can be prevented by including a per-node sequence number in the synchronization messages. Specifically, each node maintains a sequence number for itself, and keeps a copy of the most recent sequence number received from *each* of its parent nodes. In a new round of time synchronization, each node increments its sequence number and includes it in all the messages sent to its neighbor nodes. Accordingly, a node only accepts a message from a neighbor node (and update the recorded sequence number of this neighbor node) if the sequence number in the message is greater than the recorded one.

Note that we cannot use a global sequence number to prevent replay attacks. Otherwise, a malicious node that has the right keying materials may launch Denial of Service (DoS) attacks.

Resource Consumption Attacks An attacker may attempt to launch resource consumption attacks. In level discovery phase, a malicious node may make itself the children node of all its neighbors. In the synchronization phase, all its neighbor nodes will have to unicast synchronization messages to this malicious node, which is a waste of their battery power. However, such a malicious node can only force each of its neighbor nodes to transmit a few messages in each synchronization round, and thus has limited impact.

There is a potentially more serious resource consumption attack. In the synchronization phase, a malicious node may unicast synchronization messages to its neighbor nodes at any time, without receiving any synchronization message. In other words, the malicious nodes intend to start one round of synchronization without being triggered by the source node. Fortunately, a normal node sends synchronization messages only after receiving at least $2t + 1$ synchronization messages from distinct neighbors. As a result, the malicious nodes may convince its normal children nodes to request synchronization messages from other parent nodes, but will not convince them to further send synchronization messages, as long as the victim normal node has less than $2t + 1$ malicious neighbor nodes.

Wormhole Attacks Wormhole attacks are a serious threat to multi-hop wireless networks. In a wormhole attack, an attacker tunnels packets received in one part of the network over a low latency link and replays them in a different part [44]. Attackers may tunnel level discovery messages or synchronization messages through wormholes. However, such wormhole attacks do not help much in disrupting time synchronization. An attacker may try to impair the single-hop pair-wise clock difference between two nodes by introducing delays when forwarding the related messages. Fortunately, with a sender-receiver pair-wise time synchronization technique (e.g., SPS [29]), two communicating nodes can measure the message transmission time at the same time when they measure their clock difference, and thus will be able to detect the wormhole if the wormhole introduces noticeable delay. An attacker may also establish some wormholes in level discovery phase but stop them in the synchronization phase. Such attacks are subsumed by normal node failures, and can be addressed with the maintenance of the level hierarchy.

Sybil Attacks A malicious node may forge multiple identities to send messages. However, this attack can be prevented if the two communicating nodes share a unique pair-wise key and use it to authenticate the communication messages. If colluding malicious nodes can exchange their keying materials, one malicious node may impersonate other far-away malicious nodes in its local network. Such colluding malicious nodes may be detected and removed by using the techniques proposed in [73].

5.2.4 Performance Analysis

We discuss the performance of our schemes on metrics such as communication overhead, synchronization precision, and memory requirement.

Communication Overhead When considering a sensor network as a graph, in which each vertex stands for a node in the network, and each edge represents that the two vertices of the edge are neighbor nodes, we get $G = \{V, E\}$ where $|V|$ is the number of sensor nodes and $|E|$ is the number of connections between the nodes.

In the level discovery phase of level-based approach, after a node determines its level, it unicasts level discovery messages to the neighbors that have not sent level discovery messages to it.

Assuming there is no communication failures and all the nodes are included in the level hierarchy, all the edges in the graph will be covered exactly once by one level discovery message in both approaches. Thus, the overhead is $O(|E|)$. In the synchronization phase, we assume that there is no communication failures and all the nodes in the network can synchronize their clocks. Suppose there are n_1 nodes in level one. Since the nodes at levels more than 1 will receive $2t+1$ synchronization messages, the total number of messages transmitted in one round of clock synchronization can be estimated as

$$n_1 + (|V| - n_1 - 1)(2t + 1). \tag{5.1}$$

In the diffusion-based scheme, the number of messages transmitted in one round of time synchronization is the same as that in the level discovery phase of the level-based schemes, that is $O(|E|)$. Suppose each node has k neighbor nodes in average in a large dense sensor network. We have $|E| = |V| \cdot k/2$. Compared with the level-based schemes, the diffusion-based scheme has a higher communication overhead when $k \geq 2(2t+1)$. In real sensor networks, due to the message collision, the overhead in both schemes will be higher.

Synchronization Precision The synchronization precision at a node i can be measured by the clock error between node i's estimated global clock and the actual global clock (i.e., the clock of the source node) when node i adjusts its local clock. Specifically, $Error_i = |C_i + \delta_{i,S} - C_S|$, where C_i and C_S are the node i and the source node S's local clock times, respectively, and $\delta_{i,S}$ is the estimated source clock difference.

In our scheme, the major clock error is mostly caused by the clock drift between the time when the source node starts one round of time synchronization and the time when a node obtains its source clock difference. Suppose the source node S initiates one round of synchronization at time t_s and node i adjusts its clock at time t_i, where $t_i > t_S$. We denote the maximum time duration $t_i - t_S$ of all the nodes as the *synchronization time*. By [22], when the maximum clock drift of all the clocks is ρ, the maximum clock drift during $t_i - t_s$ between node S and node i is up to $2\rho(t_i - t_s)$. It seems that a sensor node may receive $2t+1$ messages sooner in the diffusion-based scheme than in the level-based scheme, since it can receive from any neighbor node in the diffusion-based scheme. However, due to the higher communication overhead in the diffusion-based scheme, there is more message collision and message retransmission. Hence, the diffusion-based scheme has a longer synchronization time and a worse synchronization precision than the level-based scheme.

Given the required precision and maximum clock drift rate, we can decide the *synchronization interval*, which is about how often the source node initiates one round of time synchronization. Suppose the maximum clock drift rate of all the sensor nodes is ρ. Given the synchronization precision δ and the required precision Δ of an application, the synchronization interval R must satisfy that $R \leq (\Delta - \delta)/\rho$.

Memory Usage Memory usage is a critical issue for resource constrained sensor nodes. In the level discovery phase, the level-based approach requires memory to record a node's level, its parent nodes, and its children nodes. To tolerate up to t malicious node among its neighbor nodes, a normal node has to have a certain amount of memory set aside for children node so that the malicious nodes cannot prevent it from having normal children nodes by consuming this memory. In the synchronization phase of both level-based and diffusion-based approaches, each node only needs to record $2t + 1$ single-hop pair-wise clock differences and $2t + 1$ source clock differences from its neighbor nodes.

5.3 Secure and Resilient Global Time Synchronization with Multiple Source Nodes

In our initial experiments, we observe that it took a long time to synchronize a large sensor network, and some nodes are usually not synchronized. Our investigation revealed that this is mostly due to message propagation and increased occurrences of message collsions. Moreover, the nodes far away from the source node may not be synchronized with a high precision due to the clock drift during the synchronization process. To reduce the synchronization time and improve the synchronization rate and the synchronization precision, we propose to deploy multiple source nodes into the network, and make sensor nodes synchronize to the nearest source node. This approach is in essence similar to the typical techniques (e.g., [82, 83, 70, 68]) for location estimation in sensor networks, where multiple anchor nodes that know their locations are deployed to help the other nodes to estimate their locations.

The multiple source nodes can also increase the robustness of the time synchronization, so that sensor nodes can get synchronized from other source nodes even if the nearest source node fails. In hostile environments, it is possible for malicious attackers to compromise a small portion of the source nodes, though the source nodes are typically better protected from attacks than the normal ones. Thus, we also extend our techniques to tolerate a certain number of malicious source nodes.

5.3.1 Extended Model

We assume all the normal source nodes are well synchronized to an external clock, for example, through GPS receivers. Suppose the IDs of the source nodes are known to all the sensor nodes. We extend the time synchronization model in Section 5.1.2 to accommodate synchronization with multiple source nodes:

1. Each node i maintains a *local clock* C_i.
2. Each node i may obtain a source clock difference δ_{i,S_j} between its local clock and the clock of a source node S_j following the model in Section 5.1.2.
3. To tolerate up to s malicious source nodes, each node i needs to obtain at least $2s + 1$ source clock differences from distinct source nodes. Node i then chooses the median of the source clock differences as its global clock difference $\delta_{i,S}$.
4. Each node i can estimate the *global clock* C_S by using its local clock and its source clock difference (i.e., $C_S = C_i + \delta_{i,S}$).

When all the source nodes are normal (i.e., $s = 0$), sensor nodes may synchronize to any source node.

5.3.2 Estimation of Hop-Count Threshold

When multiple source nodes are used for time synchronization, each node only need synchronize to the nearest $2s + 1$ source nodes. Thus, it is unnecessary to propagate the time synchronization messages for each source node to the entire network. As a result, we can significantly reduce the message propagation time and the chances of message collisions. Therefore, we propose to limit the coverage area of each source node. Specifically, we set a suitable hop-count threshold on the maximum hop count that a synchronization message can be forwarded. We certainly still need to guarantee that each sensor node can synchronize to $2s + 1$ source nodes.

In the level-based scheme, we can set the hop-count threshold by limiting the maximum level in each source node's level hierarchy. In the level discovery phase, a sensor node only chooses the neighbor nodes whose level are less than the hop-count threshold as its parent nodes. If a sensor node's level equals to the hop-count threshold, it will not send level discovery messages to its neighbor nodes. In the synchronization phase, a sensor node may send synchronization messages only if its level is less than the hop-count threshold.

In the diffusion-based scheme, we set an uppper bound threshold on the maximum hop count for the synchronization messages. We add a hop count field in the synchronization messages. When a source node initiates one round of synchronization, it sets the hop count in the messages to 0. Each sensor node only accepts a sychronization message whose hop count field is less than the threshold, and records the hop count in the message. To tolerate up to t malicious neighbor nodes, a sensor node needs to receive $2t + 1$ messages from neighbor nodes. If the maximum hop count of the $2t + 1$ messages is less than the hop-count threshold minus one, the sensor node sends synchronization messages to its neighbors from whom it did not receive a synchronization message. Otherwise, it does not send any synchronization message.

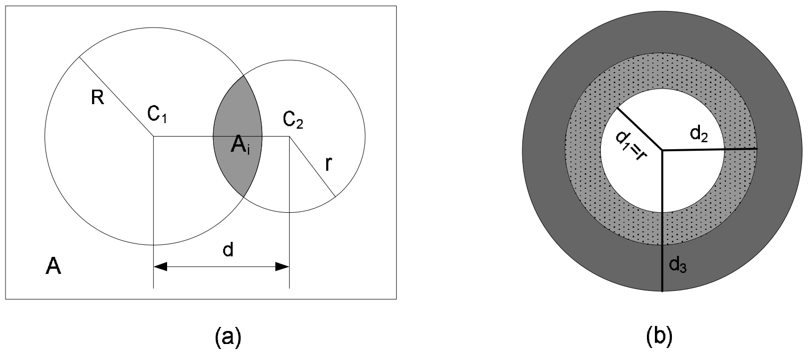


Figure 5.2: Estimation of Hop-count threshold

In the following, we present a method to derive a suitable hop-count threshold for a source node. Suppose all the nodes have the same transmission range. In Figure 5.2(a), we assume that n sensor nodes are uniformly distributed in a rectangle field of area A. Two circles C_1 and C_2 have radii R and r, respectively. The distance between the centers of the two circles is d, we can calculate the shadow area A_i of the circle intersection by:

$$A_i = r^2 \cos^{-1}(\frac{d^2 + r^2 - R^2}{2dr}) + R^2 \cos^{-1}(\frac{d^2 + R^2 - r^2}{2dR})$$
$$-\frac{1}{2}\sqrt{(r + R - d)(d + r - R)(d + R - r)(d + r + R)}. \quad (5.2)$$

Suppose a source node locates at point C_1. The sensor nodes in the circle C_1 have obtained their levels, and the maximum level of these sensor nodes is l. Consider a sensor node i that locates at point C_2 with transmission range r. In the level-based scheme, node i needs to receive from $3t+1$

neighbor nodes with less than or equal to l levels to obtain its level. This condition can be satisfied only if

$$A_i \cdot \frac{n}{A} > 3t + 1. \tag{5.3}$$

Given the node density of the network $\frac{n}{A}$, node transmission range r, and the number of malicious neighbor nodes t, by equation 5.3, we can calculate the maximum distance d from the level $l + 1$ nodes to the source node. Figure 5.2(b) shows the procedure to obtain the level threshold, given a maximum distance D from the source node. We assume that the source node locates at the center of the circle C_1, and its transmission range is r. All the level one nodes are in the transmission range of the source node, and the maximum distance from level one node to the source node is $d_1 = r$. For level two nodes, by using $R = d_1 = r$ into equation 5.2, we can get the maximum distance d_2 from the level two nodes to the source nodes. For level three nodes, by using $R = d_2$ and $r = r$ into equation 5.2, we get the maximum distance d_3 from level three nodes to the source node, and so on. Because the shadow area A_i in Figure 5.2(a) increases along with R, we have $d_{i+2} - d_{i+1} > d_{i+1} - d_i$ when $i \geq 1$, which means the bands in Figure 5.2(b) will become wider and wider. Now given the maximum distance D from the farthest node to the source node, we can calculate the a level threshold L by finding the minimum L that satisfies $d_L \geq D$. In the level-based scheme, the level threshold functions as the hop-count threshold.

In the diffusion-based approach, we can perform a similar calculation. But we should use inequation 5.4 instead of inequation 5.3, since a node needs to receive $2t + 1$ synchronization messages from neighbor nodes.

$$A_i \cdot \frac{n}{A} > 2t + 1. \tag{5.4}$$

5.3.3 Time Synchronization with Multiple Source Nodes

All source nodes are normal. When all the source nodes are normal, since a sensor node can synchronize to any source node, we can improve the synchronization performance. First, multiple source nodes make sensor nodes receive from a source node in shorter hops, so the accumulated synchronization error on the nodes along the path can be reduced. Second, multiple source nodes can reduce the message collision and shorten the synchronization time, so they can improve the synchronization precision. Moreover, the multiple source nodes may increase the synchronization rate in a randomly distributed sensor network.

In the level-based scheme, each source node builds a level hierarchy rooted by itself. For the neighbors of a source node, they choose the source node as the unique parent node. For a node more than one hop away from any source node, to tolerate up to t malicious neighbor nodes, it can choose either (1) a set of $3t + 1$ parent nodes that synchronize to the same source node, or (2) a set of $3t+1$ parent nodes that may synchronize to different source nodes. In the synchronization phase, a node may obtain a source clock difference after receiving synchronization messages from $2t + 1$ parent nodes.

In the diffusion-based scheme, the neighbors of the source nodes can synchronize their clocks after receiving from a source node. For a node more than one hop away from any source node, a node can synchronize its clock after receiving synchronization messages from any $2t + 1$ neighbor nodes.

Partial source nodes are malicious. To tolerate up to s malicious source nodes, a normal sensor node has to receive at least $2s + 1$ source clock differences from distinct source nodes.

When some source nodes are malicious, the sensor node should obtain each source clock difference from a set of parent nodes that synchronize to the same source node. Note that the sensor nodes cannot use the source clock difference obtained from a set of parent nodes that synchronize to different source nodes. Consider Figure 5.3, in which circles stand for source nodes, and triangles stand for sensor nodes. Suppose the shadow nodes are malicious. For the bottom sensor node, one of the three neighbor sensor nodes is malicious, one of the three source node is malicious, and the malicious nodes may collude with each other. In each subgraph, the bottom node obtains a source clock difference by choosing the median value of the three candidate source clock differences received from the three neighbor nodes, which synchronize to three different source nodes. Similarly, each parent node synchronizes to different source node in each subgraph. The bottom node can obtain three source clock differences from the neighbor nodes that synchronize to different source nodes, but the two source clock differences in (a) and (b) are controlled by the colluding malicious nodes. When the bottom node uses the median of the three source clock differences to synchronize its clock, it cannot correctly synchronize its clock.

In both level-based and diffusion-based schemes, the source node adds its identity into the messages that it initiates. In the level-based scheme, each source node independently builds a level hierarchy rooted at itself. When a sensor node's level in a source node's level hierarchy is no more than the level threshold, the sensor node records parent/children sets for the source node. Note that a sensor node may record parent/children sets for more than $2s + 1$ distinct source nodes. In the synchronization phase, after one sensor node obtains a source clock difference from one source

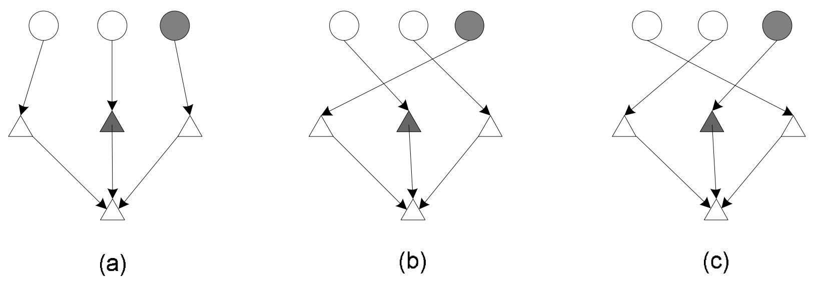


Figure 5.3: Partial malicious source nodes

node, it sends synchronization messages to the set of children nodes that synchronize to the same source node. After obtaining $2s+1$ source clock differences from different source nodes, the sensor node uses the median of the $2s + 1$ source clock differences to adjust its clock. Similarly, in the diffusion-based scheme, to tolerate s malicious source node, a sensor node synchronizes its clock after obtaining $2s + 1$ source clock differences.

When all the source nodes are normal, we can improve the performance on communication overhead and synchronization precision. When some source nodes may be malicious, we can tolerate the attacks from malicious source nodes by synchronizing sensor nodes to multiple source nodes. However, we sacrifice the performance of our schemes to achieve the robustness. To tolerate s malicious source nodes, a normal node has to obtain $2s + 1$ source clock differences from different source nodes. For each source clock difference, the node needs to receive from $2t + 1$ neighbor nodes to tolerate up to t malicious neighbor nodes. Thus, the communication overhead is increased along with s and t. To guarantee that each node can receive from $2s + 1$ source nodes, each source node has to increase its coverage area, in which message collisions increase. Due to the increased occurrences of message collisions, both the communication overhead and the synchronization time increase a lot. In the level-based scheme, each node allocates more memory to record the parent/children sets for multiple source nodes. In both level-based and diffusion-based schemes, each node needs to allocate more memory to record the candidate source clock differences from different neighbor nodes and different source nodes.

5.4 Simulation Results

We studied both level-based and diffusion-based time synchronization through simulation in ns2 [5]. Our goal is to gain a better understanding of the performance issues of the proposed techniques, which cannot be obtained through theoretical analysis. We implemented a new agent in ns2 to provide global time synchronization for sensor nodes. We used a simple "Hello" protocol for nodes to discover their neighbor nodes.

Table 5.1: Simulation parameters in level-based and diffusion-based schemes

Number of Nodes	50, 100, 150, 200
Simulation Area	60m $\times$ 60m
Transmission Range	20m
Physical Link Bandwidth	250 kbps
MAC layer	802.11 with DATA/ACK
Clock Drift Rate ($\mu s/s$)	uniformly distributed in [0, 10]
Malicious Neighbors	t = 0, 1, 3
Total Source Nodes	S = 1, 9

Malicious Source Nodes	s = 0 when S=1
	s=0, 1, 3 when S=9

Table 5.1 shows the parameters used in our experiments. The numbers of nodes n in a sensor network is 50, 100, 150, and 200, respectively, and they do not include the source nodes. All the sensor nodes remain static after being randomly deployed in a 60 m $\times$ 60 m simulation area. Suppose all the nodes have the same transmission range, which is 20 m. The bandwidth of each physical link is 250 kbps, as provided by MICAz motes [20]. Our simulation uses 802.11 with DATA/ACK as the MAC layer, in which an ACK message is sent back for a unicast DATA message, and no ACK message for broadcast DATA message. In our simulation, we did not enable the RTS/CTS/DATA/ACK pattern in the 802.11 protocol, since the control messages will introduce a large extra latency into the synchronization time and have a high collision rate on themselves. We simulate a node i's local clock as $C_i = (1 + \rho_i) \cdot C_S$, where C_S is the clock of the source node S and ρ_i is node i's clock drift rate. Each ρ_i is randomly generated according to a uniform distribution between 0 and 10 $\mu s/s$.

First, we deploy a single source node in the center of the simulation area, and assume the unique source node is always trusted. For each sensor node, the number of malicious neighbor nodes t can be 0, 1, and 3, respectively. When $t = 0$, our scheme degenerates into an existing time synchronization scheme (e.g., [31], [63]), depending on the single-hop pair-wise clock synchronization scheme adopted in our scheme. Next, we deploy 9 source nodes in the simulation area as Figure 5.4 shows. The number of malicious source nodes can be 0, 1, and 3, respectively.

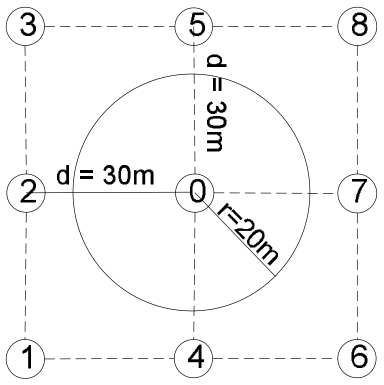


Figure 5.4: Topology of multiple source nodes in simulation

Our simulation adopts a simple single-hop pair-wise clock synchronization scheme: one-way pair-wise time synchronization [67]. Specifically, with a single source node S, a node j sends its current global clock time $C_S^{(j)}$ in the synchronization messages. After receiving this message, node i can calculate its source clock difference by $\delta_{i,S} = \delta_{i,j} + \delta_{j,S} = C_j - C_i + \delta_{j,S} = C_S^{(j)} - C_i$. The advantage of this scheme is that a node can obtain the pair-wise clock difference from a neighbor and the neighbor's source clock difference in one synchronization message, so it can decrease the possibility of message collision and shorten the synchronization time. Note that if we use TPSN [31] to obtain the single-hop pair-wise clock difference, we may get a higher precision on the pair-wise clock difference than this simple approach. However, since a node needs to exchange at least 2 messages with each parent node, the message collision will increase more than twice, the synchronization time is longer, and the synchronization precision becomes worse.

5.4.1 Single Source Node

When deploying a single source node in the network, we study the performance of our schemes when they can tolerate up to t malicious sensor nodes. We compare the level-based scheme and the diffusion-based scheme on synchronization rate, communication overhead, synchronization time, and synchronization precision. Each data point in the result figures is an average of 10 simulation runs with identical configuration but different randomly generated node deployments. The Y axis error bars show confidence interval at 95% confidence.

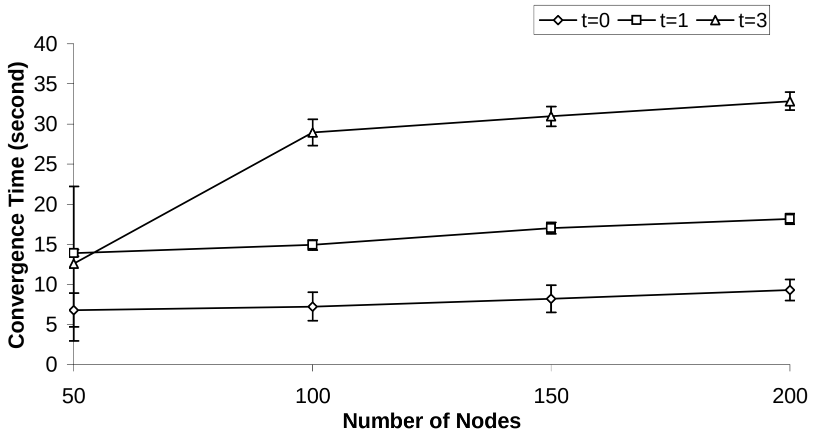


Figure 5.5: Convergence time of level discovery phase

Convergence Time of Level Discovery The convergence time of the level discovery phase is shown in Figure 5.5. When $n = 50$ and $t = 3$, the level hierarchy cannot include all the sensor nodes. In our simulation, to reduce a node's level in the level hierarchy, after obtaining its level, each sensor node delays 1 second before it sends level discovery messages to its neighbor nodes.

Synchronization Rate Figure 5.6 shows the percentage of sensor nodes that can be synchronized. When $t = 3$ and $n = 50$, due to the relatively low density of the network, the level-based scheme can synchronize only 40% nodes, while diffusion-based scheme can synchronize 60% nodes. When n increases to 150, both schemes can synchronize almost all the sensor nodes. The diffusion-based scheme can synchronize more sensor nodes than the level-based scheme in the sparse sensor networks. When $t = 3$ and $n = 200$, due to the increased message collision, several sensor nodes may not be synchronized in the level-based scheme.

Communication overhead In both schemes, the neighbors of the source node require only one synchronization message from the source node, and the nodes more than one hop away from the source node need to receive at least $2t + 1$ messages from neighbor nodes.

Figure 5.7 shows the number of synchronization message sent in one round of time synchronization. One message can be retransmitted at most 4 times in our simulation. The diffusion-based approach has a higher communication overhead than the level-based approach. The communication overheads increase along with the number of the sensor nodes in both schemes. In the

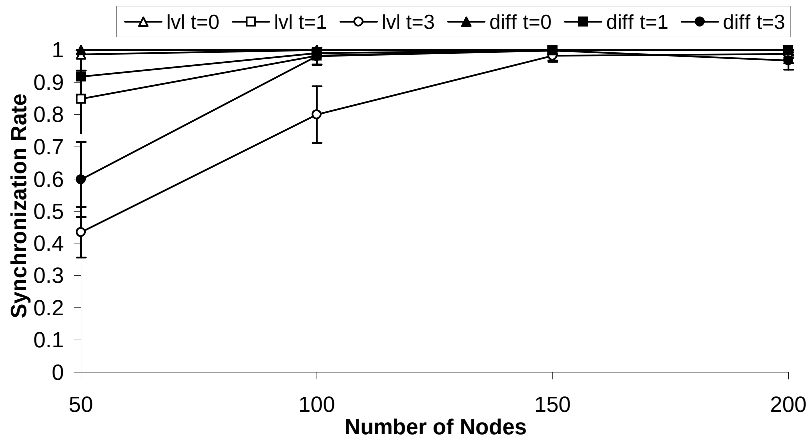


Figure 5.6: Synchronization rate using unicast

level-based scheme, the overhead also increases along with t, since nodes need to receive from more parent nodes. In the diffusion-based scheme, when t is smaller, a node will send to more neighbor nodes from whom it has not received the message. It may receive from a neighbor node before it sends out the buffered message to the neighbor node, but it will resume to send the message in our simulation. Thus, in the diffusion-based scheme, the communication overhead decreases a little when t increases.

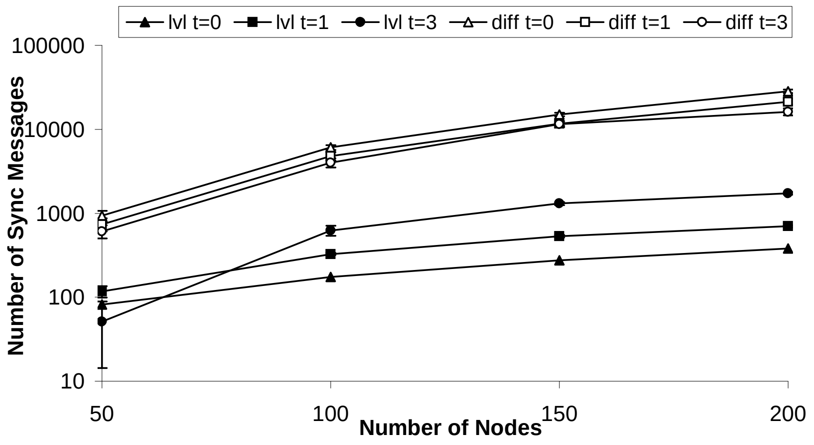


Figure 5.7: One round communication overhead using unicast

Synchronization Time Before measuring the synchronization precision, we first examine the synchronization time, which has a major effect on the synchronization precision.

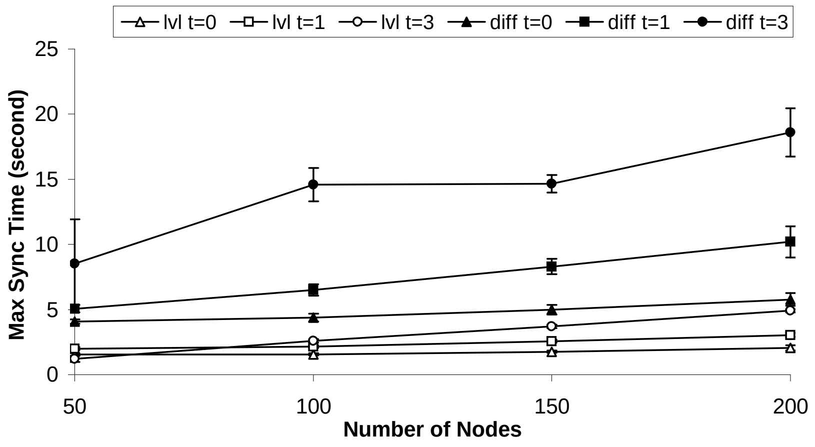


Figure 5.8: Maximum synchronization time using unicast

Figure 5.8 and 5.9 show the maximum and average synchronization times to finish one round of global time synchronization. The synchronization time increases along with t in both schemes. The level-based scheme can finish sooner than the diffusion-based scheme. The level-based scheme can finish one round of synchronization in 4 seconds when $t = 3$; while in the diffusion-based scheme, the maximum synchronization time is near 16 seconds when $t = 3$.

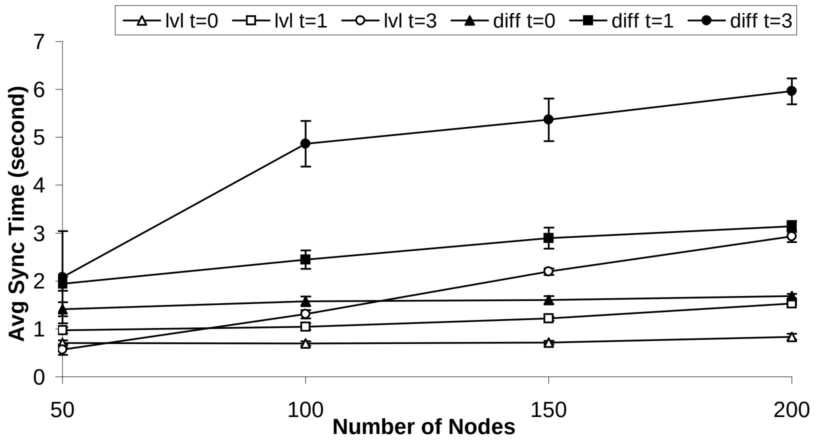


Figure 5.9: Average synchronization time using unicast

Synchronization Precision Now we examine the synchronization precision of the proposed schemes. To reduce the impact of the nondeterministic delays caused by the MAC layer, we follow [31] to use MAC layer timestamp. We modify the 802.11 MAC layer in ns2 to record the exact timestamp when a message is transmitted or received physically.

Figure 5.10 and 5.11 show the maximum and average clock errors of all the nodes immediately after synchronizing their local clocks. The level-based scheme can provide a much better clock precision than the diffusion-based scheme. The major reason is that the clock drift during the synchronization time is greater in the diffusion-based scheme than that in the level-based scheme.

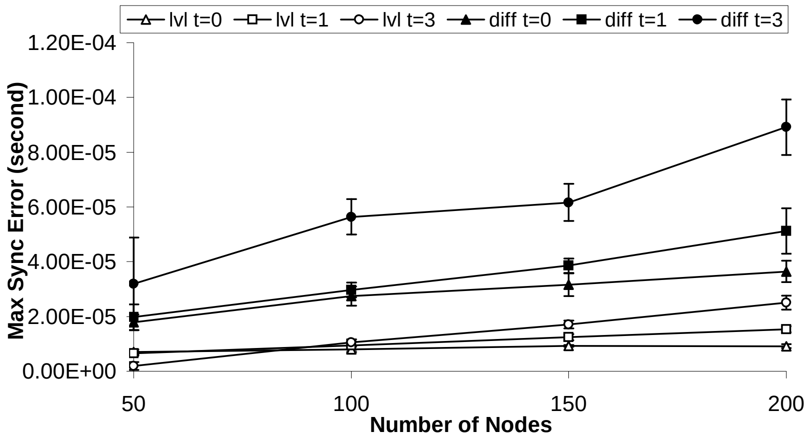


Figure 5.10: Maximum synchronization error using unicast

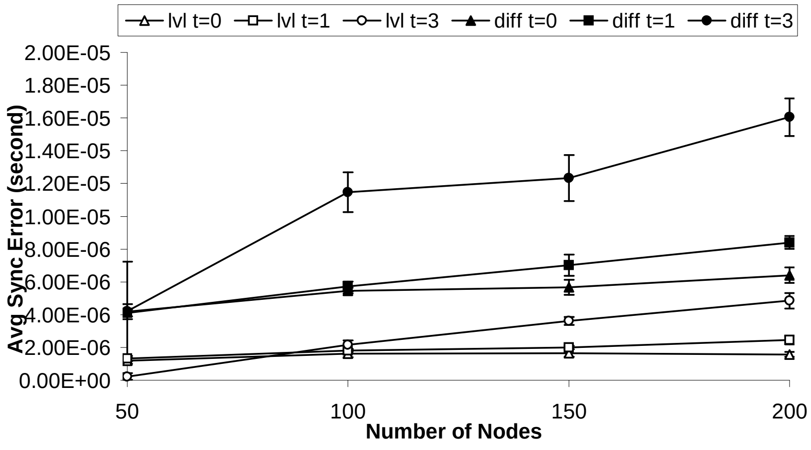


Figure 5.11: Average synchronization error using unicast

5.4.2 Multiple Source Nodes

We deploy 9 source nodes in the network, as Figure 5.4 shows. When up to s source nodes may be compromised, a sensor node has to obtain source clock differences from $2s+1$ source nodes.

In our simulation, we fix the number of sensor nodes to 200, and both t and s can take values from 0, 1, and 3.

Hop-count Threshold First, we need to decide the hop-count threshold to allow all the sensor nodes receive from $2s+1$ source nodes. When $s=0$, all the source nodes are normal. Since the multiple source nodes can guarantee that all the sensor nodes can directly receive from at least one source node, the hop-count threshold is set to 1. When $s=1$, each sensor node needs to receive source clock differences from at least 3 source nodes. As Figure 5.12(a) shows, the maximum distance for all the sensor nodes to receive from the nearest 3 source nodes is $D = 15 * \sqrt{5} = 33.54m$. When $s=3$, each node needs to receive from 7 source nodes. Figure 5.12(b) shows that the maximum distance for a sensor node to receive from the nearest 7 source nodes is $D = 30 * \sqrt{5} = 67.08m$. Table 5.2 shows the hop-count thresholds when $n = 200$, calculated by following Section 5.3.2.

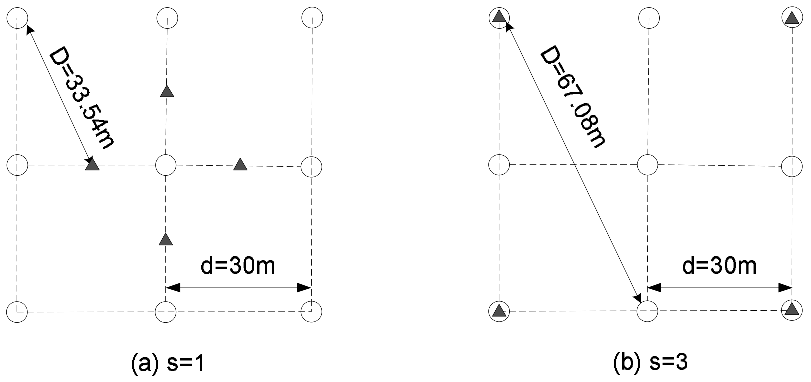


Figure 5.12: Maximum distance from 2s+1 source nodes

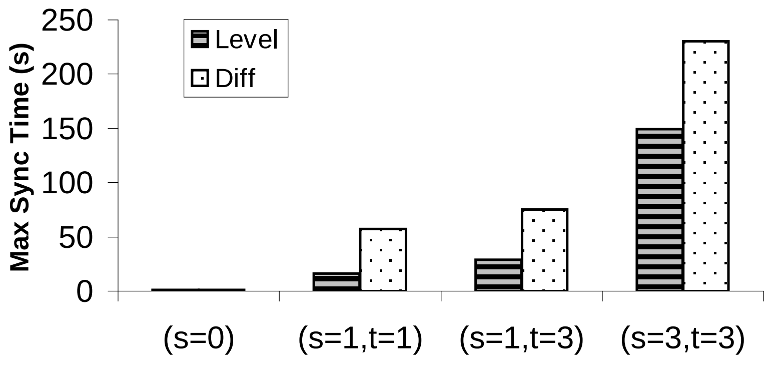
Table 5.2: Hop-count thresholds when n=200, S=9.

Scheme	s=0	s=1, t=1	s=1, t=3	s=3, t=3
Level-based	1	2	3	6
Diffusion-based	1	2	3	5

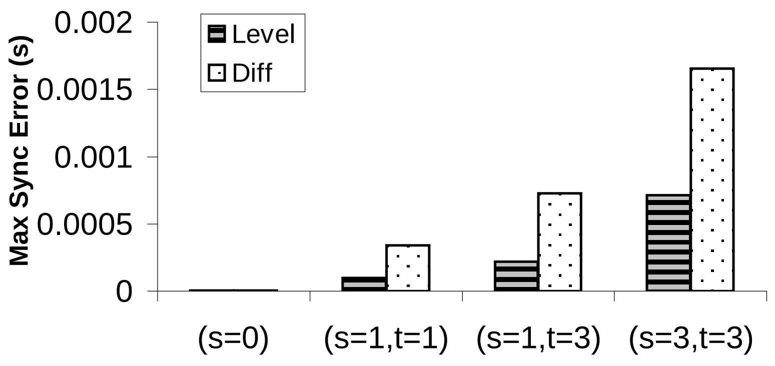
In the level-based scenarios, the source nodes can build their level hierarchies at different times to prevent the message collision caused by messages from other source nodes.

Synchronization Rate In all the scenarios, all the 200 sensor nodes can be synchronized.

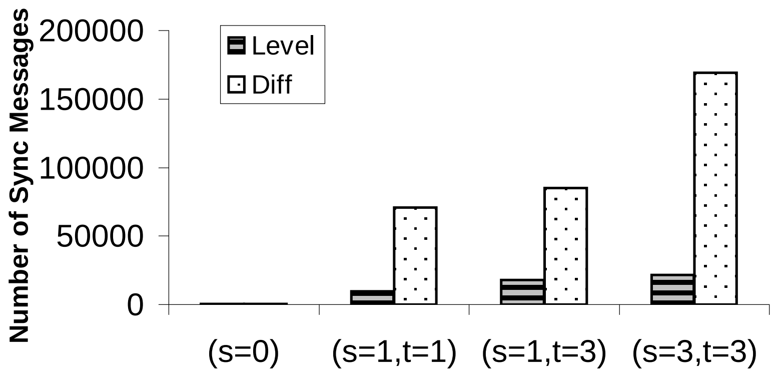
Synchronization Time From our simulation results, when $s=0$ and $s=1$, all the source nodes may initiate the synchronization process at the same time. Due to the small hop-count thresholds, the message collision is under control. However, when $s=3$, if all the source



(a) Maximum Synchronization Time



(b) Maximum Synchronization Error



(c) Communication Overhead

Figure 5.13: Experimental results with multiple source nodes

nodes synchronize at the same time, due to the high hop-count thresholds, there is a huge message collision which makes it impossible to synchronize the sensor nodes. To reduce the collision, we divide the 9 source nodes in to 5 groups, that is, $\{1, 8\}$, $\{2, 6\}$, $\{3, 7\}$, $\{4, 5\}$ and $\{0\}$. The source nodes in the same group can initiate synchronization at the same time, since they are relatively far from each other and have less message collision. In our simulation, each group initiates the synchronization process in an interval of 20 seconds in the level-based scheme, and an interval of 30 seconds in diffusion-based scheme. This arrangement will increase the synchronization time and the synchronization error, but maximizes the synchronization rate. There may exist better ways to arrange the order for source nodes to initiate the time synchronization, and we consider it as our future work.

Figure 5.13(a) shows the maximum synchronization time in different scenarios. We can see that the synchronization time increases along with the t and s. When $s = 0$, the whole network can be synchronized in one second, no matter the value of t, since all the sensor nodes can be directly synchronized.

When $s = 1$ and $t = 1$, the synchronization time is around 16 seconds in the level-based scheme, and around 57 seconds in the diffusion-based scheme. When $s = 1$ and $t = 3$, because a node far away from a source node needs to receive 7 clock differences before sending its synchronization messages, the time increases to around 29 seconds in the level-based scheme, and around 75 seconds in the diffusion-based scheme. When $s = 3$ and $t = 3$, the level-based scheme needs around 2.5 minutes to finish one round of synchronization, while the diffusion-based scheme needs almost 4 minutes. In order to decrease the synchronization time, we may distributed more source nodes into the network.

Synchronization Error Figure 5.13(b) shows the maximum synchronization error. When $s = 0$, the maximum synchronization error is less than 10 μs. When $s = 1$ and $t = 3$, the maximum synchronization error is less than 0.23 ms in the level-based scheme, and 0.6 ms in the diffusion-based scheme. When $s = 3$ and $t = 3$, the maximum synchronization error increases almost 3 times.

Communication Overhead When $s = 0$, the message overheads in both schemes are less than 400. The communication overhead in the level-based scheme is moderate for sensor nodes. When $s = 3$ and $t = 3$, in average, each sensor node sends nearly 100 messages in one round of synchronization. The communication overhead of the diffusion-based scheme is much higher than the level-based scheme. When $s = 1$ and $t = 1$, each node has to send nearly 200 messages. When $s = 3$ and $t = 3$, each node sends around 850 messages. Considering the resource constraint in sensor nodes, it makes the diffusion-based scheme inscalable to tolerate more malicious source nodes.

5.5 Summary

In this chapter, we presented two secure and resilient global time synchronization schemes for sensor networks. We adopted a model where all the sensor nodes synchronize their clocks to a common source, which is assumed to be well synchronized to an external clock. We propose to increase the performance by deploying multiple source nodes, and extend our schemes to tolerate malicious source nodes.

When we developed these two schemes, we believed that the broadcast authentication is not applicable in wireless sensor networks. Therefore, we chose to provide unicast authentication by using the secret pair-wise key shared between two communicating nodes, and we can guarantee the security of our schemes. However, because each node needs to send one message to each neighbor node in each round of time synchronization, the communication overhead is quite high, and it may cause potential huge message collisions. Therefore, our schemes are difficult to be used in large sensor networks.

In our consecutive research, we developed a secure and resilient global time synchronization using broadcast authentication based on a novel use of the μTESLA broadcast authentication protocol for *local authenticated broadcast*, resolving the conflict between the goal of achieving time synchronization with μTESLA-based broadcast authentication and the fact that μTESLA requires loose time synchronization. In each round of global synchronization, each node only broadcasts one synchronization message. We will discuss the detail in next chapter.

Chapter 6

Secure and Resilient Global Time Synchronization with Broadcast Authentication

In this chapter, we describe the design of *TinySeRSync* [94], a secure and resilient global time synchronization using broadcast authentication. The protocol is based on a novel way to integrate broadcast authentication into time synchronization, which successfully provides authentication of the source, the content, and the timeliness of synchronization messages.

We implement TinySeRSync on MICAz motes running TinyOS and perform a thorough evaluation through field experiments in a network of 60 MICAz motes. The evaluation results indicate that TinySeRSync is a practical system for secure and resilient time synchronization in wireless sensor networks.

6.1 TinySeRSync: Secure and Resilient Global Time Synchronization

In this section, we propose a secure and resilient global time synchronization protocol, TinySeRSync, which is integrated with local broadcast authentication. We adopt the same global time synchronization model in Section 5.1. The source node broadcasts synchronization messages periodically to adjust the clocks of all sensor nodes. The synchronization messages are propagated throughout the network to reach nodes that cannot communicate with the source node directly. The timely transmission of all these messages are authenticated. Moreover, each node obtains synchronization information from multiple neighbor nodes, so that it can tolerate compromised nodes to a certain extent.

6.1.1 Basic Approach

To deal with the ad hoc deployments of sensor networks and the lack of initial synchronization among sensor nodes, TinySeRSync consists of two *asynchronous* phases: *Phase I–secure single-hop pair-wise synchronization*, and *Phase II–secure and resilient global synchronization*. In Phase I, pairs of neighbor nodes exchange messages with each other to obtain single-hop pair-wise time synchronization. Phase I uses authenticated MAC layer timestamping and a two message exchange to ensure the authentication of the source, the content, and the timeliness of synchronization messages. Nodes run Phase I periodically to compensate (continuous) clock drifts and maintain certain pair-wise synchronization precision, providing the foundation for global time synchronization as well as the μTESLA-based local broadcast authentication in Phase II.

Phase II uses *authenticated local (re)broadcast* to achieve global time synchronization, starting with a broadcast synchronization message from the source node. Phase II adapts μTESLA to ensure the timeliness and the authenticity of the local broadcast synchronization messages. To be resilient against potential compromised nodes, each node estimates multiple candidates of the global clock using synchronization messages received from multiple neighbor nodes, and chooses the median. Nodes that are synchronized to the source node further rebroadcast the synchronization messages locally. This process continues until all the nodes are synchronized. Phase II also runs periodically to maintain certain global time synchronization precision.

We would like to emphasize that the two phases are *asynchronous*. In other words, secure single-hop pair-wise synchronization (Phase I) is executed by nodes individually and independently, while secure and resilient global synchronization (Phase II) is controlled by the source node and propagated throughout the network. The only requirement is that a node finishes Phase I before entering Phase II. Also note that both Phase I and Phase II are executed periodically. Though a node that has not performed Phase I synchronization with its neighbor nodes cannot participate in a global synchronization, it may join the next round of global synchronization once it finishes Phase I. Thus, our approach supports incremental deployment of sensor nodes, which is an important property required by many sensor network applications.

TinySeRSync has a critical difference from the schemes proposed in Chapter 5: it uses *authenticated local broadcast* to propagate global synchronization messages, while the schemes in Chapter 5 uses authenticated unicast that leads to substantial communication overhead as well as message collisions. This difference represents a key step that enables practical secure and resilient time synchronization in sensor networks.

The ability to authenticate local broadcast messages is the cornerstone of the proposed protocol. In the following, we describe in detail how this is done in TinySeRSync.

6.1.2 Authentication of Local Broadcast Synchronization Messages

As discussed earlier, the signaling messages for global time synchronization are broadcast in nature, and must be transmitted in a timely and authenticated way. There are two general solutions for authenticating broadcast messages in sensor networks: digital signatures and μTESLA [74, 76]. Though it is possible to verify digital signatures on sensor platforms, as shown in [37], digital signature operations are still multiple order of magnitude more expensive than secret key based solutions such as μTESLA. Using digital signatures for time synchronization may quickly exhaust the battery power of sensor nodes. Moreover, it is also an attractive target of Denial of Service (DoS) attacks: An attacker may broadcast synchronization messages with false digital signatures to force sensor nodes to perform expensive signature verifications.

μTESLA [76] relies on symmetric cryptography, and thus does not suffer from the above problems. However, μTESLA requires loose time synchronization between the broadcast sender and the receivers. Considering the goal of having the source node synchronize the clocks of all the sensor nodes, there seems to be a conflict in using μTESLA for authenticating broadcast time synchronization messages.

We can indeed avoid the above conflict. We observe that two neighbor nodes may securely perform single-hop pair-wise time synchronization using the techniques in Chapter 3. Consider an arbitrary node A. Assume node A have synchronized with all its neighbor nodes so that node A and any of its neighbor nodes know the clock difference between them. As a result, if node A needs to broadcast a synchronization message to all its neighbor nodes, it may certainly use μTESLA for broadcast authentication, since the "loose synchronization" requirement needed by μTESLA is already satisfied. In other words, we only use μTESLA *locally* to avoid the above conflict.

Specifically, we adapt μTESLA for local broadcast authentication to protect the broadcast messages from a node to its neighbors, assuming the Phase I neighbor synchronization has completed. In the following, we first give a brief introduction to μTESLA, and then discuss the adaptation of μTESLA in TinySeRSync.

Overview of μTESLA Protocol

An asymmetric mechanism such as public key cryptography is generally required for broadcast authentication [74]. Otherwise, a malicious receiver can easily forge any message from the sender, as discussed earlier. μTESLA introduces asymmetry by delaying the disclosure of symmetric keys [76]. A sender broadcasts a message with a MIC generated with a secret key K, which is disclosed after a certain period of time. When a receiver gets this message, if it can ensure that the message was sent before the key was disclosed, the receiver buffers this message and authenticates the message when it later receives the disclosed key. To continuously authenticate broadcast messages, μTESLA divides the time period for broadcast into multiple intervals, assigning different keys to different time intervals. All messages broadcast in a particular time interval are authenticated with the key assigned to that time interval.

To authenticate the broadcast messages, a receiver first authenticates the disclosed keys. μTESLA uses a one-way key chain for this purpose. The sender selects a random value K_n as the last key in the key chain and repeatedly performs a (cryptographic) hash function F to compute all the other keys: $K_i = F(K_{i+1}), 0 \leq i \leq n-1$, where the secret key K_i (except for K_0) is assigned to the i-th time interval. Because of the one-way property of the hash function, given K_j in the key chain, anybody can compute all the previous keys $K_i, 0 \leq i \leq j$, but nobody can compute any of the later ones $K_i, j+1 \leq i \leq n$. Thus, with the knowledge of the initial key K_0, which is called the *commitment* of the key chain, a receiver can authenticate any key in the key chain by merely performing hash function operations. When a broadcast message is available in the i-th time interval, the sender generates a MIC for this message with a key derived from K_i, broadcasts this message along with its MIC, and discloses the key K_{i-d} for time interval I_{i-d} in the broadcast message (where d is the disclosure lag of the authentication keys). Figure 6.1 illustrates the division of the time line and the assignment of authentication keys in μTESLA.

Each key in the key chain will be disclosed after some delay. As a result, the attacker can forge a broadcast message by using the disclosed key. μTESLA uses a security condition to prevent such situations. When a receiver receives an incoming broadcast message in time interval

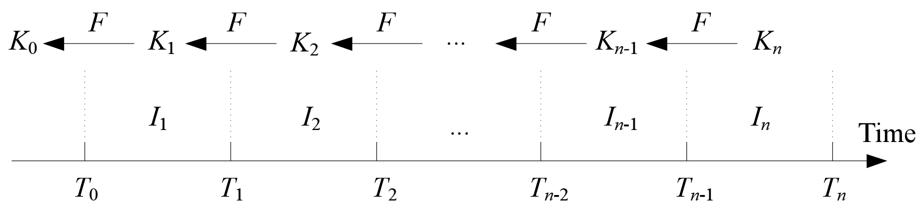


Figure 6.1: μTESLA protocol

I_i, it checks the security condition $\lfloor(T_c + \Delta - T_1)/T_{int}\rfloor < i + d - 1$, where T_c is the local time when the message is received, T_1 is the start time of the time interval 1, T_{int} is the duration of each time interval, and Δ is the maximum clock difference between the sender and itself. If the security condition is satisfied, i.e., the sender has not disclosed the key K_i yet, the receiver accepts this message. Otherwise, the receiver simply drops it.

Short Delayed μTESLA: Adapting μTESLA for Global Time Synchronization

Distribution of μTESLA Parameters: In order to use μTESLA, the sender needs to transmit a number of parameters to all the receivers before the actual broadcast messages. These include the key chain ID, the key chain commitment, the duration of each time interval, and the starting time of the first time interval. We can fix the duration of time intervals and the length of each key chain as network wide parameters. However, the other parameters have to be communicated from each node to its neighbors. To reduce communication cost, we piggy-back the transmission of these μTESLA parameters with the single-hop pair-wise synchronization between neighbors. In other words, each node sends the parameters of its own μTESLA key chain to a neighbor node during secure single-hop pair-wise synchronization. When one key chain is about to expire, each node needs to communicate with each neighbor node again to transmit the parameters for the next key chain.

Balancing Key Chain Size and Authentication Delay: A direct application of μTESLA to authenticate the local broadcast synchronization messages faces a risk. μTESLA is subject to DoS attacks [75], in which an attacker overhearing a valid broadcast message may use the disclosed key in the message to forge broadcast synchronization messages. A receiver has to buffer all such (forged) messages claimed to be from some neighbor until it receives the disclosed key. As a result, the receiver may not have enough memory to buffer synchronization messages from other neighbor

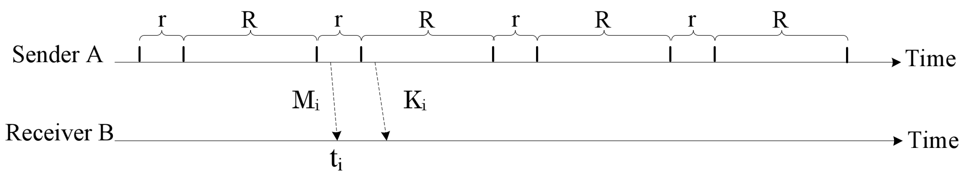


Figure 6.2: Short delayed μTESLA protocol

nodes. The immediate authentication mechanism proposed in [75] cannot be applied here, because it requires that the sender know the next message to be transmitted before sending the current message.

One possible way to mitigate the threat of DoS attacks in global synchronization is to exploit the tight time synchronization established during Phase I. Specifically, when using μTESLA for local broadcast authentication, we may use very short time intervals to limit the duration vulnerable to DoS attacks. Because the neighbor nodes have been tightly synchronized with each other during phase I, the broadcast sender can use very short time intervals and disclose an authentication key right after the corresponding interval is over. When the time interval is short enough, it does not give enough time to an attacker to forge broadcast messages using the disclosed key it just learns from the valid broadcast message. A short enough interval duration also offers authentication of the *timeliness* of the synchronization messages; it disallows a replayed message to be transmitted in the valid time interval, and thus enables receivers to detect and remove them.

However, this approach comes with a significant cost: To cover a certain period of time (e.g., 30 minutes), the sender needs to generate a fairly long key chain due to the short time intervals, and most of the keys will be wasted. Reducing the key chain length will force all the neighbor nodes to exchange the key chain commitments frequently, leading to heavy communication overhead.

We propose to adapt μTESLA to address the above conflict. Specifically, we proposed to use two different intervals in one μTESLA instance, a short interval r and a long interval R. The short intervals and the long intervals are interleaved, as shown in Figure 6.2. As in the original μTESLA, each time interval is still associated with an authentication key, which is used to authenticate messages sent in this time interval. Each node broadcasts a message authenticated with μTESLA only during the short intervals, while broadcasting the disclosed key in the following long interval (possibly multiple times to tolerate message losses).

Upon receiving a broadcast message, a receiver first checks the security condition using the (MAC layer) message receipt time. Because each receiver and the sender have synchronized tightly with each other, the receiver can easily transform the receipt time into the time point in the sender's clock, and verify if the corresponding authentication key has been disclosed when the receiver receives the message. Consider Figure 6.2. Suppose the receiver B receives a synchronization message M_i from the sender A at its local time t_i (taken in the MAC layer), and the start time of A's μTESLA instance is T_0 in A's clock. B may calculate $i = \lfloor \frac{t_i - T_0 + \Delta_{B,A}}{r+R} \rfloor$ and checks the following security condition: $t_i - T_0 + \Delta_{B,A} + \delta_{max} < i * (R + r) + r$, where $\Delta_{B,A}$ is the pair-wise clock difference between A and B, and δ_{max} maximum synchronization error between two neighbor nodes. B stores the message and the number i only if this check is successful. Otherwise, B simply drops the message. After node B obtains the disclosed key K_i, it verifies $F^{i-j}(K_i) = K_j$ with a previously received key or commitment K_j where $j < i$. If the key is valid, B then uses K_i to verify the MIC included in the broadcast synchronization message M_i.

6.2 Analysis

6.2.1 Security Analysis

Phase I. Phase I provides authentication of the source and the content of synchronization messages. Moreover, Phase I uses a two-way message exchange to estimate both the clock difference between direct neighbors and the transmission delay, and can detect attacks that attempt to mislead time synchronization by introducing extra message delays. Thus, Phase I provides protection of the source, the content, and the timeliness of single-hop pair-wise synchronization messages. Specifically, Phase I effectively defeats external attacks that attempt to mislead single-hop pair-wise time synchronization, including forged and modified messages, pulse-delay attacks, and wormhole attacks that introduce extra delays. Phase I protocol cannot handle DoS attacks that completely jam the communication channel. Nevertheless, no existing protocol can survive such extreme DoS attacks.

Phase II. Given the secure single-hop pair-wise synchronization protocol in Phase I, the remaining threats to global time synchronization are two-fold. First, an external attacker may fake or replay (local) broadcast messages used for global synchronization to mislead the regular nodes. To defend against this threat, Phase II adapts μTESLA to provide local broadcast authentication. The security of this variation of μTESLA follows directly from the original scheme [74]. Besides local broadcast authentication, another benefit of using μTESLA is the authentication of the timeliness of local broadcast synchronization messages, since a delayed message will be automatically discarded due to the violation of the security condition. Thus, similar to Phase I, by authenticating the source, the content, and the timeliness of local broadcast synchronization messages, Phase II can successfully defeat all the external attacks that are intended to mislead the time synchronization.

Second, a compromised node may provide misleading synchronization information to disrupt the global time synchronization. Thus, our global time synchronization protocol must be resilient to compromised nodes. Since the source node is trusted, in Phase II, each direct neighbor node of the source node can directly estimate the global clock securely. However, the other nodes may receive false synchronization information from compromised nodes. The solution used by Phase II is to have each node use the source clock differences received from $2t + 1$ neighbor nodes to estimate $2t + 1$ candidate source clock differences, and select the median one as its own source clock difference. It is easy to see that if every node has no more than t compromised neighbor nodes, Phase II can successfully synchronize all the normal nodes as long as they have enough number of neighbor nodes. Similar to Phase I, Phase II cannot handle DoS attacks that completely jam the communication channel.

In conclusion, TinySeRSync provides a practical solution to provide secure and resilient global time synchronization in wireless sensor networks. It can successfully defeat all non-DoS external attacks against time synchronization, and is resilient to compromised nodes.

6.2.2 Performance Analysis

Synchronization Precision and Coverage. TinySeRSync uses predication-based MAC layer timestamping in Phase I, avoiding many places that could introduce uncertainty during time synchronization. In Phase II, TinySeRSync tries to estimate the global clock through the estimation of source clock differences, and thus greatly reduces the impact generated by the propagation delays of synchronization messages. Thus, TinySeRSync can provide high precision time synchronization. Moreover, TinySeRSync employs flooding-based propagation of global synchronization messages; this allows all the nodes that have enough number of neighbor nodes to be synchronized.

Communication, Computation, and Storage Overheads. TinySeRSync uses message exchanges between direct neighbor nodes for Phase I synchronization. All these message exchanges are local, and do not introduce wide area interference. In Phase II, TinySeRSync adopts local broadcast for the propagation of global synchronization messages, effectively harnessing the broadcast nature of wireless communication. Thus, TinySeRSync is efficient in terms of communication.

TinySeRSync uses efficient symmetric cryptography for message authentication. In particular, it exploits the hardware cryptographic support provided by the CC2420 radio component. Thus, TinySeRSync introduces very light computation overhead for cryptographic operations.

TinySeRSync does increase the storage overhead on sensor nodes due to the need to maintain cryptographic keys, buffer the local broadcast messages, and store the source clock differences received from $2t + 1$ neighbor nodes. A critical issue is the maintenance of the μTESLA key chain required for authenticating outgoing synchronization messages. Our adaptation of μTESLA greatly reduces the number of keys in each key chain. In addition, we use another approach to further reduce the memory requirement and the delay: After generating a key chain, each node only saves some select keys called *key anchors* (e.g., 1 of every 10 keys), and also caches the keys before the next key anchor to be used (e.g., the first 10 keys). When a μTESLA key is required for authentication, if the key is available in the cache, the node can directly use it. Otherwise, the node can regenerate and fill the key cache using the next key anchor.

Incremental Deployment. As discussed earlier, TinySeRSync uses two asynchronous phases, both of which are executed periodically. Thus, TinySeRSync works well with incremental deployment of sensor nodes. The newly deployed nodes first obtain the pair-wise time differences and the commitments of the key chains from its neighbor nodes in Phase I, and then join the Phase II global time synchronization. Our experimental results in Section 6.3 will show the performance when there are incrementally deployed sensor nodes.

6.3 Experiment Results

Our implementation of TinySeRSync is targeted at MICAz motes [4]. However, our implementation can be used with slight modification for other sensor platforms that also use CC2420 radio components, such as TelosB [7] and Tmote Sky [8]. MICAz has an 8-bit micro-controller *ATMega128L* [1], which has 128k Byte program memory and 4k Byte SRAM. As discussed earlier, MICAz is equipped with the ChipCon CC2420 radio component [6], which works at 2.4 GHz radio frequency and provides up to 250 kbps data rate. CC2420 is an IEEE 802.15.4 compliant RF transceiver that features hardware security support.

We performed a series of experiments in a network of 60 MICAz motes to evaluate the performance of TinySeRSync in real deployment. We focused on the performance metrics in normal situations, while relying on the analysis in Section 6.2 for the security properties.

6.3.1 Configuration

Figure 6.3 shows the sensor network test-bed used in our experiments. The different node shapes represent nodes deployed at different times during incremental deployment, which will be explained in Section 6.3.3. The test-bed consists of 60 nodes, among which node 1 (with the solid circle) is configured as the source node.

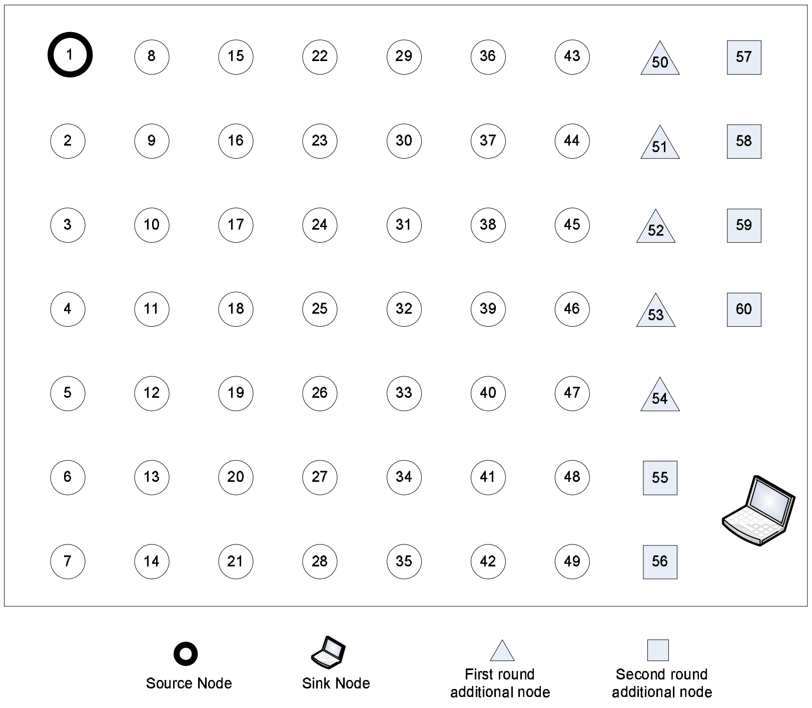


Figure 6.3: Deployment of network test-bed.

We use a number of parameters in our evaluation. Each node performs a secure single-hop pair-wise synchronization with its neighbor nodes for every $d_1 = 4$ seconds. During this synchronization, the node informs its neighbor nodes its μTESLA parameters. The source node starts a global synchronization every d_2 seconds. In our experiments, we use $d_2 =$ 5 or 10 seconds. The degree of tolerance (against compromised neighbor nodes) is represented as t, as used throughout this dissertation. In our experiments, we use $t = 0, 1, 2, 3, 4$ to examine the various performance metrics.

We use a sink node to help collecting data from each sensor node. Periodically, the sink node broadcasts an anchor message with the highest power to all the nodes. Upon receiving this message, each node marks the receiving time and converts it to the global time using its source

Table 6.1: Code size of TinySeRSync on MICAz motes

Memory	Size (bytes)
RAM	1961
ROM	24814

clock difference. The sink node then queries each node individually to get the receiving time (in the estimated global clock) along with other auxiliary information. This allows us to discover the synchronization error on each individual sensor node, the synchronization rate, as well as the number of synchronization levels each node has to go through.

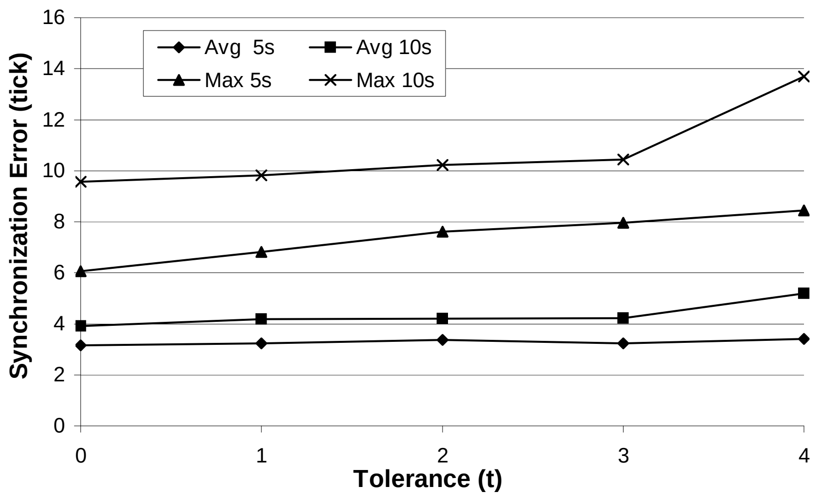
Let us first look at the code size of TinySeRSync on TinyOS [41] before presenting the performance results. The code size is related to the maximum number of neighbor nodes each node may have. For each neighbor node, a node will spend 46 bytes to save the pair-wise key, current key in key chain, and clock differences, etc. In our experiments, each node saves 10 keys for a key chain with 100 keys. Each node reserves a buffer to store at most 6 unauthenticated global synchronization messages, which increase the size of RAM. Table 6.1 shows the code size of TinySeRSync when each node may have at most 8 neighbor nodes. The RAM size will increase to 3137 bytes to accommodate 36 neighbor nodes.

6.3.2 Performance in Static Deployment

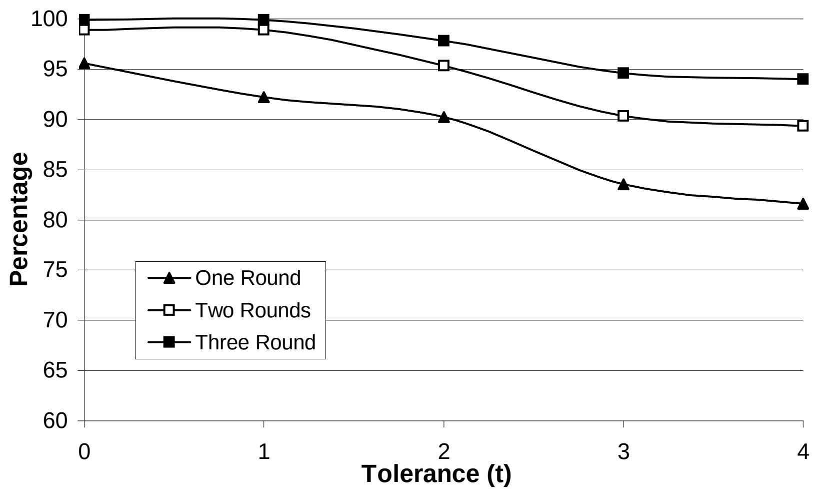
Let us first look at the performance of TinySeRSync in static deployments. In our experiments, we use the following metrics to evaluate the performance and the overhead of TinySeRSync: the average and the maximum synchronization errors, the synchronization rate (i.e., the percentage of nodes that can be synchronized), the synchronization level (i.e., the maximum number of hops that global synchronization messages have to go through before a sensor node can be synchronized.

Average and Maximum Synchronization Error: Figure 6.4(a) shows the maximum and the average synchronization error with different global synchronization intervals and different degrees of tolerance against compromised neighbor nodes. In all the cases, the maximum synchronization error is below 14 ticks (121.52 μs), and the average synchronization error is below 6 ticks (52.08 μs). Figure 6.4(a) also indicates that as the global synchronization interval increases, the maximum and the average synchronization errors both increase.

Synchronization Rate: Figure 6.4(b) shows the synchronization rate (i.e., the percentage of nodes that can be synchronized by TinySeRSync) after one, two, and three rounds of global synchro-



(a) Maximum and average synchronization error

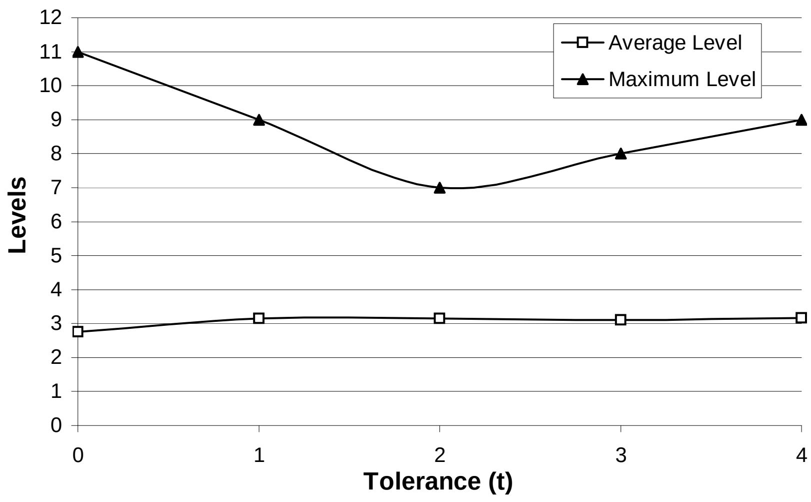


(b) Synchronization rate

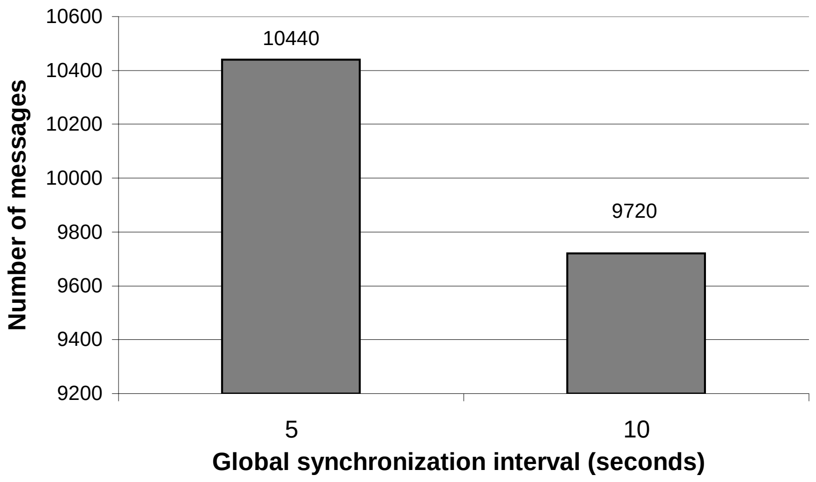
Figure 6.4: Synchronization error and synchronization rate using broadcast

nization. When the tolerance against compromised neighbor nodes increases, as we expected, the synchronization rate decreases. However, after three rounds of global synchronization, even in the worst case, about 95% of the nodes can be synchronized to the source node.

Synchronization Level: Figure 6.5(a) shows the maximum and the average number of hops the global synchronization messages have to traverse before all the nodes are synchronized. In our test-bed, in all cases, the average synchronization level is around 3. An interesting issue is that



(a) Maximum and average synchronization level



(b) Communication overhead (# messages each node sends per hour)

Figure 6.5: Synchronization level and communication overhead using broadcast

the maximum synchronization level initially decreases as the tolerance t increases, but then goes up as t is greater than 2. This is because when t is very small (i.e., $t = 0, 1$), a node can broadcast the synchronization message almost immediately after it is synchronized. The synchronization triggered by these fast nodes may be propagated to many nodes that have not been synchronized. However, when t is large enough, synchronizing a node with increased t requires receiving synchronization messages from more neighbor nodes, thus resulting in an increasing trend for maximum synchronization levels.

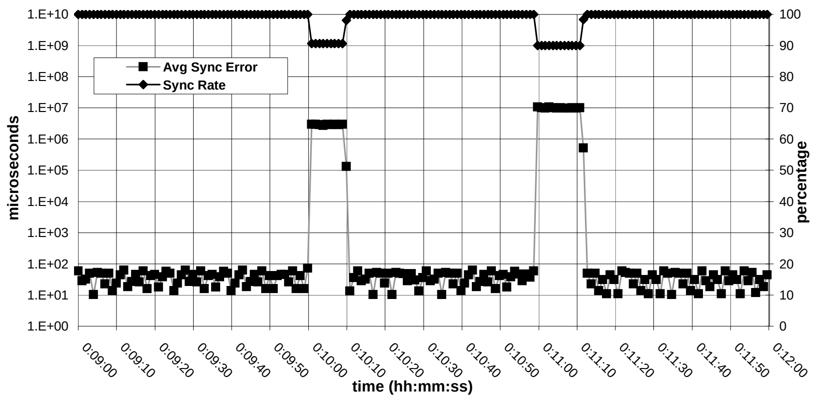
Communication Overhead: We measure the communication overhead by assessing the number of messages each node has to transmit per time unit. For each neighbor node, a node sends one message to obtain the pair-wise time difference. In one round of global time synchronization, each node at most broadcasts one synchronization message and one key disclosure message. Suppose each node has n neighbor nodes, the pair-wise synchronization interval is d_1, and the global synchronization interval is d_2. In a given long time interval T, each node sends at most $n \cdot \frac{T}{d_1} + \frac{2T}{d_2}$ messages. Figure 6.5(b) shows the communication overhead per hour for a configuration where each node has 10 neighbor nodes, the pair-wise time synchronization interval is 4 seconds, and the global time synchronization interval is 10 seconds.

6.3.3 Incremental Deployment

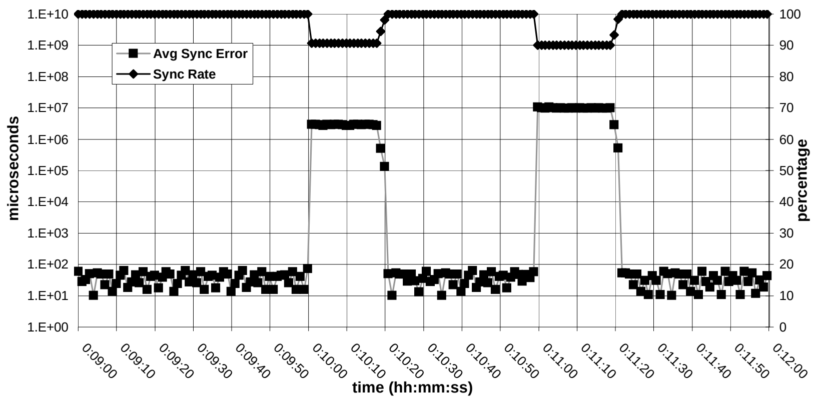
We evaluated the performance of TinySeRSync when there were incremental deployments. Consider Figure 6.3. At the beginning of the experiment, we deployed the 49 nodes marked as circles. We then added 5 new nodes into the network about 10 minutes later, and added another 6 new nodes about 1 minute later. In this experiment, we set the global synchronization interval as 10s. Figure 6.6 shows the history of the average synchronization error and the coverage in this experiment, when t=0, 2, and 4. As shown in the figure, when the new nodes were just added into the network, they could not be synchronized immediately, and the average synchronization error was large and the synchronization rate dropped to around 90%. However, after a few rounds of global synchronization, all these new nodes were correctly synchronized, resulting in a low average synchronization error and 100% synchronization coverage.

6.4 Summary

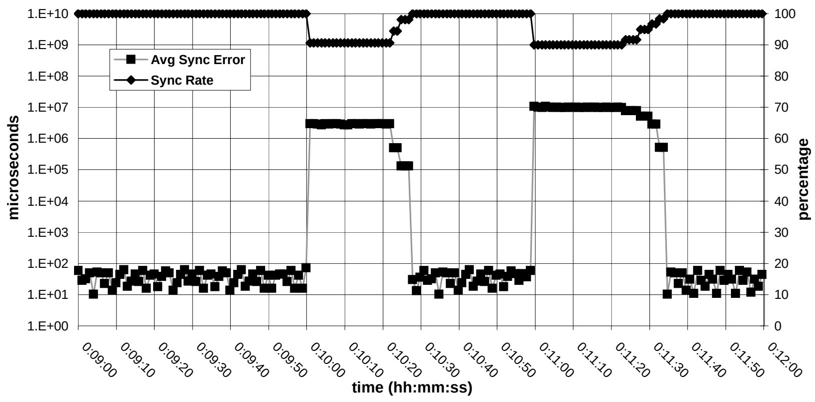
In this chapter, we develop a secure and resilient time synchronization protocol called *TinySeRSync* for wireless sensor networks, targeting common sensor platforms such as MICAz and TelosB running TinyOS [41]. Our protocol offers a novel way to integrate (broadcast) authentication into time synchronization, which successfully provides authentication of the source, the content, and the timeliness of synchronization messages. Our novel use of μTESLA in secure global time synchronization successfully resolved the conflict between the goal of achieving time synchronization



(a) t=0



(b) t=2



(c) t=4

Figure 6.6: Average synchronization error (left Y-axis) and coverage (right Y-axis) during incremental deployment

and the fact that μTESLA requires loose time synchronization. The resulting protocol is secure against external attacks and resilient against compromised nodes.

We provide an implementation of the proposed techniques on TinyOS and a thorough evaluation through field experiments in a network of 60 MICAz motes. The evaluation results indicate that TinySeRSync is a practical system for secure and resilient global time synchronization in wireless sensor networks.

Chapter 7

Conclusions

This dissertation contains a suite of techniques to achieve secure time synchronization between two neighbor nodes, among a group of sensor nodes, and in a whole sensor network, respectively.

- *Secure Single-hop Pair-wise Time Synchronization:* We develop a secure single-hop pair-wise time synchronization protocol by using a *hardware-assisted, authenticated MAC layer timestamping* technique to handle high data rate such as those produced by MICAz and TelosB motes. With the hardware security support in radio components, we implement the proposed technique on MICAz motes [20] running TinyOS [41]. The secure single-hop pair-wise time synchronization serves as the building block to achieve the secure and resilient global time synchronization.

- *Fault-tolerant Cluster-Wise Time Synchronization:* This technique provides a novel fault-tolerant cluster-wise clock synchronization for a cluster of sensor nodes, where the nodes in each cluster can communicate with each other directly through broadcast. In each round of time synchronization, only one node serves as the *synchronizer*, and only one authenticated synchronization message is broadcast. Thus, our scheme can avoid the message collision problem. The proposed scheme exploits a recently proposed local broadcast authentication technique for sensor networks, which is purely based on symmetric cryptography [104], thus avoiding the costly digital signature for message authentication. Our analysis shows that the proposed scheme guarantees an upper bound on the clock difference between nonfaulty nodes when no more than $1/3$ of the nodes are compromised and collude with each other. We propose a secure distributed cluster formation algorithm to divide the sensor networks into mutual disjoint cliques, which are required by the fault-tolerant cluster-wise time synchronization.

- *Secure and Resilient Global Time Synchronization:* This research resulted in two secure and resilient time synchronization schemes: level-based and diffusion-based time synchronization. The level-based scheme builds a level hierarchy in the wireless sensor network, and then synchronizes the whole network level by level. The diffusion-based scheme allows each node to diffuse its clock to its neighbor nodes after it has synchronized to the source node. To improve the performance and the resilience of our techniques, we propose to deploy multiple source nodes in the network.

  We first propose to use authenticated unicast messages to distribute the synchronization information by using secure key shared between each two neighbor nodes. However, due to high communication overhead and huge message collisions, it can hardly be used in large sensor networks. To solve this problem, we develop a secure and resilient global time synchronization, TinySeRSync, based on a novel use of the μTESLA broadcast authentication protocol for *local authenticated broadcast*, resolving the conflict between the goal of achieving time synchronization with μTESLA-based broadcast authentication and the fact that μTESLA requires loose time synchronization. We implement TinySeRSync on TinyOS and a thorough evaluation through field experiments in a network of 60 MICAz motes. The evaluation results indicate that TinySeRSync is a practical system for secure and resilient global time synchronization in wireless sensor networks.

In my future work, I plan to investigate additional techniques that can improve the synchronization precision in our time synchronization techniques. For example, we can adapt the linear regression technique proposed in [63] to compensate the constant clock drifts. Because linear regression technique requires each node store a time vector in its RAM for each neighbor node, we must allocate the RAM memory carefully. TinySeRSync is the first secure and resilient global time synchronization implemented in real wireless sensor networks. We will look into the integration of TinySeRSync in sensor network applications, such as target tracking, data fusion, and power saving.

Bibliography

[1] ATmega128(L) Complete Technical Documents. `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

[2] dfmax.c. `ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/`.

[3] GPS time transfer. `http://tf.nist.gov/timefreq/time/gps.htm`.

[4] Micaz: Wireless measurement system. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.

[5] The network simulator – ns-2. `http://www.isi.edu/nsnam/ns/`.

[6] SmartRF CC2420 Datasheet (rev 1.3), 2005-10-03. `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`.

[7] Telosb mote platform. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`.

[8] Tmote sky: Reliable low-power wireless sensor networking eases development and deployment. `http://www.moteiv.com/products-tmotesky.php`.

[9] IEEE standard 802.11. wireless lan medium access control (MAC) and physical layer (PHY) specification, 1999.

[10] A.D. Amis, R. Prakash, T.H.P. Vuong, and D. T. Huynh. Max-Min D-cluster formation in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, March 1999.

[11] D.J. Baker, A. Ephremides, and J.A. Flynn. The design and simulation of a mobile radio networkwith distributed control. *IEEE Journal on Selected Areas in Communications*, SAC-2(1):226–237, 1984.

[12] S. Bandyopadhyay and E. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proceedings of IEEE INFOCOM*, 2003.

[13] B. Barak, S. Halevi, A. Herzberg, and D. Naor. Clock synchronization with faults and recoveries. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 133–142, 2000.

[14] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, 1999.

[15] K. Chakrabarty, S. S. Iyengar, H. Qi, and E. Cho. Grid coverage for surveillance and target location in distributed sensor networks. *IEEE Transactions on Computers*, 51:1448–1453, 2002.

[16] H. Chan and A. Perrig. ACE: An emergent algorithm for highly uniform cluster formation. In *European Workshop on Wireless Sensor Networks (EWSN 2004)*, Jan 2004.

[17] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Research in Security and Privacy*, pages 197–213, 2003.

[18] M. Chatterjee, S.K. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2):193–204, 2002.

[19] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[20] Crossbow Technology Inc. Wireless sensor networks. `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`.

[21] H. Dai and R. Han. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(1):125–139, 2004.

[22] D. Dolev, J. Y. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

[23] D. Dolev and H.R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal of Computing*, 12(4):656–665, 1983.

[24] J. R. Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Mar 2002.

[25] W. Du, J. Deng, Y. S. Han, and P. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 42–51, October 2003.

[26] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36:147–163, 2002.

[27] J. Elson and K. Römer. Wireless sensor networks: A new regime for time synchronization. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, pages 149–154, October 2002.

[28] A. Galleni and D. Powell. Consensus and membership in synchronous and asynchronous distributed systems. Technical Report 96104, LAAS, April 1996.

[29] S. Ganeriwal, S. Capkun, C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *Proceedings of 2005 ACM Workshop on Wireless Security (WiSe 2005)*, pages 97–106, September 2005.

[30] S. Ganeriwal, D. Ganesan, H. Shim, V. Tsiatsis, and M. B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, pages 130–141, 2005.

[31] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 138–149, 2003.

[32] M. R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman And Company, 1979.

[33] M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[34] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Dependable Computing for Critical Applications–5*, volume 10 of *Dependable Computing and Fault Tolerant Systems*, pages 139–157, Champaign, IL, sep 1995.

[35] J. Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 11–19, September 2003.

[36] N. Gura, A. Patel, and A. Wander. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, pages 119–132, August 2004.

[37] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, August 2004.

[38] R. Gusella and S. Zatti. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions on Software Engineering*, 15(7):847–853, 1989.

[39] J.Y. Halpern, B.B. Simons, H.R. Strong, and D. Dolev. Fault-tolerant clock synchronization. In *Proceedings of Third Annual ACM Symposium on Principles of Distributed Computing*, pages 89–102, 1984.

[40] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences HICSS*, 2000.

[41] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[42] A. Hu and S. D. Servetto. Asymptotically optimal time synchronization in dense sensor networks. In *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 1–10, September 2003.

[43] Y. Hu, A. Perrig, and D. B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, Dec 2001.

[44] Y.C. Hu, A. Perrig, and D.B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of INFOCOM 2003*, April 2003.

[45] L. Huang and T. H. Lai. On the scalability of ieee 802.11 ad hoc networks. In *Proceedings of the 3th ACM international symposium on Mobile ad hoc networking and computing MobiHoc '02*, 2002.

[46] IEEE Computer Society. IEEE 802.15.4: Ieee standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks – specific requirements part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). `http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf`, October 2003.

[47] H. Ishii and H Kakugawan. A self-stabilizing algorithm for finding cliques in distributed systems. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Oct 2002.

[48] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 33–42, August 2001.

[49] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*, November 2004.

[50] C.M. Krishna, K.G. Shin, and R.W. Butler. Ensuring fault tolerance of phase-locked clocks. *IEEE Transactions on Computers*, 34(8):752–756, 1985.

[51] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *SIGCOMM Computer Communication Review*, 27(2), 1997.

[52] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[53] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[54] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[55] Q. Li and D. Rus. Global clock synchronization in sensor networks. In *Proceedings of IEEE INFOCOM 2004*, pages 214–226, March 2004.

[56] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 52–61, October 2003.

[57] D. Liu, P. Ning, and R. Li. TinyKeyMan: Key management for sensor networks. `http://discovery.csc.ncsu.edu/software/TinyKeyMan/`.

[58] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, 1984.

[59] J. Lundelius-Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[60] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG:a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[61] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *Proceedings of IEEE Conference on Sensor and Ad hoc Communications and Networks SECON*, October 2004.

[62] M. Manzo, T. Roosta, and S. Sastry. Time synchronization attacks in sensor networks. In *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 107–116, 2005.

[63] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 39–49, Nov 2004.

[64] L. Meier, P. Blum, and L. Thiele. Internal synchronization of drift-constraint clocks in ad-hoc sensor networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing MobiHoc '04*, 2004.

[65] D.L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[66] S. Mishra and A. Nasipuri. An adaptive low power reservation based mac protocol for wireless sensor. In *Proceedings of the IEEE International Conference on Performance Computing and Communications*, pages 713–736, 2004.

[67] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Clock synchronization for wireless local area networks. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (Euromicro-RTS 2000)*, June 2000.

[68] A. Nasipuri and K. Li. A directionality based location discovery scheme for wireless sensor networks. In *Proceedings of ACM WSNA'02*, pages 105–111, September 2002.

[69] J. Newsome, R. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: Analysis and defenses. In *Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN 2004)*, April 2004.

[70] D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AoA. In *Proceedings of IEEE INFOCOM 2003*, pages 1734–1743, April 2003.

[71] A. Olson and K.G. Shin. Fault-tolerant clock synchronization in large multicomputer systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):912–923, 1994.

[72] S. PalChaudhuri, A.K. Saha, and D.B. Johnson. Adaptive clock synchronization in sensor networks. In *Information Processing in Sensor Networks (IPSN)*, pages 340–348, April 2004.

[73] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *IEEE Symposium on Security and Privacy*, May 2005.

[74] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, May 2000.

[75] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of Network and Distributed System Security Symposium*, February 2001.

[76] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks*, pages 521–534, July 2001.

[77] P. Ramanathan, D.D. Kandlur, and K.G. Shin. Hardware-assisted software clock synchronization for homogeneous distributed systems. *IEEE Transactions on Computers*, 39(4):514–524, 1990.

[78] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, 1990.

[79] K. Römer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182, 2001.

[80] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[81] K. Römer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, *Wireless Sensor Networks*. John Wiley Sons, 2005. To appear.

[82] A. Savvides, C. Han, and M. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of ACM MobiCom '01*, pages 166–179, July 2001.

[83] A. Savvides, H. Park, and M. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *Proceedings of ACM WSNA '02*, pages 112–121, September 2002.

[84] F.B. Schneider. A paradigm for reliable clock synchronization. Technical Report TR 86–735, Cornell University, Department of Computer Science, 1986.

[85] F.B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report TR 87–859, Cornell University, Department of Computer Science, 1987.

[86] K.G. Shin and P. Ramanathan. Clock synchronization of a large multiprocessor system in the presence of malicious faults. *IEEE Transactions on Computers*, 36(1):2–12, 1987.

[87] M.L. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *IEEE Wireless Communications and Networking Conference WCNC03*, 2003.

[88] H. Song, S. Zhu, and G. Cao. Attack-resilient time synchronization for wireless sensor networks. In *Proceedings of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'05)*, 2005.

[89] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[90] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 13(2), 2005.

[91] K. Sun, , P. Peng, P. Ning, and C. Wang. Secure distributed cluster formation in wireless sensor networks. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC 22)*, December 2006.

[92] K. Sun, P. Ning, and C. Wang. Fault-tolerant cluster-wise clock synchronization for wireless sensor networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(3):177–189, July–September 2005.

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

[94] K. Sun, P. Ning, C. Wang, A. liu, and Y. Zhou. Tinysersync: Secure and resilient time synchronization in wireless sensor networks. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS'06)*, pages 42–51, November 2006.

[95] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *First ACM International Workshop on Wireless Sensor Networks and Applications WSNA02*, pages 32–41, September 2002.

[96] N. Vasanthavada and P.N. Marinos. Synchronization of fault-tolerant clocks in the presence of malicious failures. *IEEE Transactions on Computers*, 37(4):440–448, 1988.

[97] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus. TinyPK: Securing sensor networks with public key technology. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04)*, October 2004.

[98] B. H. Wellenhoff, H. Lichtenegger, and J. Collins. *Global Positions System: Theory and Practice*. Springer Verlag, 4th edition, 1997.

[99] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom ’01: Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001.

[100] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[101] C.D. Young and J. A. Stevens. Clique activation multiple access (cama): A distributed heuristic for building wireless datagram networks. In *Proceedings of Military Communications Conference MILCOM*, 1998.

[102] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proceedings of IEEE INFOCOM*, March 2004.

[103] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the 1st International Workshop on Sensor Network Protocols and Applications*, May 2003.

[104] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS’03)*, pages 62–72, October 2003.