Fault-Tolerant Cluster-Wise Clock Synchronization for Wireless Sensor Networks

Kun Sun, Peng Ning, *Member*, *IEEE*, and Cliff Wang, *Member*, *IEEE*

Abstract—Wireless sensor networks have received a lot of attention recently due to their wide applications, such as target tracking, environment monitoring, and scientific exploration in dangerous environments. It is usually necessary to have a cluster of sensor nodes share a common view of a local clock time, so that all these nodes can coordinate in some important applications, such as time slotted MAC protocols, power-saving protocols with sleep/listen modes, etc. However, all the clock synchronization techniques proposed for sensor networks assume benign environments; they cannot survive malicious attacks in hostile environments. Fault-tolerant clock synchronization techniques are potential candidates to address this problem. However, existing approaches are all resource consuming and suffer from message collisions in most of cases. This paper presents a novel fault-tolerant clock synchronization scheme for clusters of nodes in 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 clock synchronization approaches, the proposed technique does not introduce collisions between synchronization messages, nor does it require costly digital signatures.

Index Terms—Clock synchronization, wireless sensor networks, fault tolerance.

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. Sensor nodes are typically resource constrained and usually communicate with each other through wireless links.

An accurate and synchronized clock time is crucial in many sensor network applications, particularly due to the collaborative nature of sensor networks. For example, in target tracking applications, 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 (e.g., [3], [39]). Sensor nodes usually contain inexpensive oscillators with typical clock drift rates at about tens of microseconds per second [33], and the clock drift (more than 1 second apart per day) is intolerable for most sensor network applications. Therefore, clock synchronization becomes indispensable for such applications. However, due to the resource constraints on sensor nodes, traditional clock synchronization protocols (e.g., NTP [25]) cannot be directly applied in sensor networks.

Several clock synchronization protocols (e.g., [10], [13], [21], [26], [37], [29], [16]) have been proposed for sensor networks to achieve either *pair-wise clock synchronization* or *global clock synchronization*. Pair-wise clock synchronization aims to obtain a high-precision clock synchronization between pairs of neighbor nodes, while global clock synchronization aims to provide network-wide clock synchronization in a sensor network. Existing pair-wise clock synchronization protocols use either *receiver-receiver synchronization* (e.g., RBS [10]), in which a reference node broadcasts a reference packet to help pairs of receivers identify the clock differences, or *sender-receiver synchronization* (e.g., TPSN [13]), where a sender communicates with a receiver to estimate the clock difference. Most of the global clock synchronization protocols (e.g., [10], [13], [37]) establish multihop paths in a sensor network so that all the nodes can synchronize their clocks to a given source based on these paths and the pair-wise clock differences between adjacent nodes in these paths. Alternatively, diffusion-based global synchronization protocols [21] achieve global synchronization by spreading local synchronization information to the entire network.

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 [41]. 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

- *K. Sun and P. Ning are with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695. E-mail: {ksun3, pning}@ncsu.edu.*
- *C. Wang is with the US Army Research Office, PO Box 12211, Research Triangle Park, NC 27709. E-mail: cliff.wang@us.army.mil.*

Manuscript received 11 Sept. 2004; revised 15 Apr. 2005; accepted 6 May 2005; published online 2 Sept. 2005.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0133-0904.

TABLE 1
Notations

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

of nodes. Indeed, none of the aforementioned clock synchronization protocols can survive malicious actions by compromised nodes. A compromised node may disrupt the clock synchronization by sending different time to noncompromised nodes. For example, when RBS [10] is used for pair-wise synchronization, a compromised node may provide different noncompromised nodes different time values about the receipt of the reference packet.

It is natural to consider fault-tolerant clock synchronization techniques, which have been studied extensively in the context of distributed systems (e.g., [31], [19], [15], [7], [23], [24], [38], [34], [35], [28], [2], [18], [36], [40]). However, traditional fault-tolerant clock synchronization techniques are not directly applicable to sensor networks. These techniques were developed for distributed systems that do not have the same resource constraints as 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 [7], CSM [19]) or multiple copies of messages (e.g., COM, CNV [19]) 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 sensor networks. Even when digital signature is used, for example, in HSSD [7], 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 [7], ST [38]) 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 sensor networks. The approach in [28] divides nodes into overlapping clusters and runs one of the previous algorithms (e.g., CNV [19], LL [23], [24]) within each cluster. This reduces the communication overhead, but cannot avoid message collisions in sensor networks.

In this paper, 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. In each round of clock 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 [42], 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.

The rest of this paper is organized as follows: Section 2 describes a model for fault-tolerant clock synchronization. Section 3 presents our fault-tolerant cluster-wise clock synchronization scheme. Section 4 discusses related work. Section 5 concludes this paper and points out some future research directions.

2 Fault-Tolerant Clock Synchronization Model

In this section, we describe our model for fault-tolerant clock synchronization in sensor networks, which is adapted from [7]. We first model the clocks on sensor nodes, and then present the properties that a fault-tolerant clock synchronization protocol should satisfy. For readers' convenience, Table 1 lists the notations used in this paper.

We 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 of sensor nodes. We use lowercase letters to denote the variables and constants about real time and uppercase letters to denote those about clock time. A 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. 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 $t_1 < t_2$:

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

The rate of drift between any two well-behaved clocks is bounded by $\lambda = \rho(2+\rho)/(1+\rho)$, which is less than 2ρ. Sensor nodes usually contain inexpensive crystal oscillators, and the typical clock drift rate is tens of microseconds [33].

A sensor node is *nonfaulty* if it correctly executes a given clock synchronization algorithm and its clock is well-behaved. 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 fth 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.,

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

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,

$$C_i(t_2) - C_j^+(t_2) = C_i(t_2) - C_i(t_1) < \epsilon, \quad (3)$$

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.,

$$|C_i(t_0) - C_j(t_0)| < \delta_0. \quad (4)$$

In the next section, we develop a new fault-tolerant cluster-wise clock 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, $|C^+(t) - C(t)| \leq k\Delta$.

3 Fault-Tolerant Cluster-Wise Clock Synchronization

3.1 Overview

In this paper, 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 [22], [4], [9].) 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 [8], where a malicious node impersonates multiple nodes by claiming multiple IDs. Fortunately, recent studies [27] 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 clock synchronization scheme executes once for every R time units. For convenience, we call such an 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 [24] to achieve initial clock synchronization, and adapt algorithm *OM* [20] 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 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

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 [20]. 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.

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 nonsynchronizer 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 clock synchronization.)

Our scheme works under the *arbitrary attack model* [12], 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 clock synchronization schemes.

3.2 Broadcast Authentication

In each round of clock 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 [42] to authenticate the broadcast synchronization messages. First, each node generates a one-way key chain $\{K^{(0)}, K^{(1)}, \ldots, K^{(l)}\}$ in the following way: $K^{(i-1)} = F(K^{(i)}), (1 \le i \le l)$, 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 [17] 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.

3.3 Fault-Tolerant Clock Synchronization Algorithm

The proposed scheme executes one round of clock 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 clock 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 fth 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 fth round.

In the second task, when a node receives a synchronization message at its clock time T in the fth round of clock 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 $k\Delta \le \bar{\Delta} \le x$, it increases its clock time by $k\Delta$; if $-x \le \bar{\Delta} \le -k\Delta$, 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 clock 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 nonfaulty and have been maintaining synchronized clocks, then there must exist two nonfaulty 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 1 (Fig. 1) shows the pseudocode. 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 [19], 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 1. *After a nonfaulty node i adjusts its clock to a nonfaulty synchronizer s's clock at t_i^f, where $beg^f \le t_i^f \le end^f$, 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 (3), 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 (1), we have

$$(C_i(t) - C_s(t)) - (C_i^+(t_i^f) - C_s(t_i^f)) \le |(C_i(t) - C_i^+(t_i^f)) - (C_s(t) - C_s(t_i^f))| < 2\rho(t - t_i^f) < 2\rho\epsilon.$$

By (3), 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 1. *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 $f \ge 0$, $|C_i(end^f) - C_j(end^f)| < \epsilon(1 + 4\rho)$. 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 $beg^{f+1} \le t_i^{f+1} \le t_j^{f+1} \le end^{f+1}$. We consider three time intervals separated by t_i^{f+1} and t_j^{f+1} in $[end^f, end^{f+1}]$.

For any $t \in [end^f, t_i^{f+1})$, by (1) and (2), 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 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 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 $f \ge 0$.

According to the above proof, we can see that for all $f \ge 0$ 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. □

Algorithm 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 ($Order(N_i) = f \bmod n$) **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 ($Order(N_i) = f \bmod n$)

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

Fig. 1. Algorithm 1. Synchronizer Ring.

The maximum clock difference $\delta + \epsilon(1+6\rho)$ can only be reached between two nonsynchronizer 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 2. *If the fth ($f \geq 1$) 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 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 antennae 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 3. *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 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 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. *Suppose during $[beg^{f+1}, end^{f+1}]$, the maximum clock difference $\bar{\Delta}$ is between node i and node j. If $\bar{\Delta} \leq (2km+1)\Delta + m\delta$ and the $(f+1)$th synchronizer is nonfaulty, for any $t \in [end^{f+1}, beg^{f+2}]$,*

$$|C_i(t) - C_j(t)| \leq MAX(\bar{\Delta} - (k-1)\Delta, \Delta).$$

Proof. Suppose node i and j adjust clocks at time t_i and t_j. If $\bar{\Delta} < k\Delta$, by Lemma 3, for any $t \in [end^{f+1}, beg^{f+2}]$,

$$\begin{aligned}|C_i(t) - C_j(t)| &< |C_i(end^{f+1}) - C_j(end^{f+1})| \\ &+ 2\rho(t - end^{f+1}) < \epsilon(1+4\rho) + \delta = \Delta.\end{aligned}$$

When $k\Delta \leq \bar{\Delta} \leq (2km+1)\Delta + m\delta$, if node i is the synchronizer, according to our algorithm, node j adjusts its clock with $k\Delta$ at t_j. We have

$$\begin{aligned}|C_i(end^{f+1}) - C_j(end^{f+1})| &< |C_i(t_j) - C_j^+(t_j)| \\ &+ 2\rho(end^{f+1} - t_j) < (\bar{\Delta} - k\Delta + \epsilon) + 2\rho\epsilon.\end{aligned}$$

So, for any $t \in [end^{f+1}, beg^{f+2}]$, by (2), we have

$$\begin{aligned}|C_i(t) - C_j(t)| &< |C_i(end^{f+1}) - C_j(end^{f+1})| + 2\rho(t - end^{f+1}) \\ &< \bar{\Delta} - k\Delta + \epsilon(1+2\rho) + \delta < \bar{\Delta} - (k-1)\Delta.\end{aligned}$$

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 $|C_i(t_i) - C_s(t_i)| \geq k\Delta$ or $|C_j(t_j) - C_s(t_j)| \geq k\Delta$, 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 3, 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 $t \in [end^{f+1}, beg^{f+2}]$,

$$|C_i(t) - C_j(t)| \leq MAX(\bar{\Delta} - (k-1)\Delta, \Delta).$$

□

Lemma 5. *When $n > 3m$, Algorithm 1 satisfies the following conditions: 1) For all $f \geq 1$ and $t \in [beg^f, beg^{f+1}]$, given any two nonfaulty nodes i and j,*

$$|C_i(t) - C_j(t)| \leq (2km+1)\Delta + m\delta + 2\rho\epsilon.$$

2) If a node makes an adjustment to its clock at time t, then $|C^+(t) - C(t)| \leq k\Delta$.

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

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

$$|C_i(t) - C_j(t)| \leq (2km+1)\Delta + m\delta + 2\rho\epsilon.$$

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 1, $\delta + \epsilon(1+6\rho)$ is the maximum clock difference if all the synchronizers are nonfaulty. By Lemma 2, 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 increased the maximum clock difference by more than $2k\Delta + \delta$. Suppose it served as r_1th and r_2th synchronizer, where $r_2 = r_1 + n \leq f$. According to our hypothesis, for $t \in [beg^{r_1}, beg^{r_1+1}]$, $|C_i(t) - C_j(t)| \leq (2km+1)\Delta + m\delta + 2\rho\epsilon$. Within $[beg^{r_1}, beg^{r_2}]$, all the $n-m$ nonfaulty nodes have served as the synchronizer at least once. By Lemma 4, 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 proven that for $t \in [beg^f, beg^{f+1}]$

$$|C_i(t) - C_j(t)| \leq (2km+1)\Delta + m\delta + 2\rho\epsilon.$$

□

Based on Lemma 5 and Algorithm 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 1 is a fault-tolerant clock synchronization algorithm when $n > 3m$. Next, we show this is indeed the case.

Lemma 6. *The synchronization interval is bounded. That is, for all $f \geq 1$, $beg^{f+1} - beg^f < y$ and $end^{f+1} - end^f < y$, where $y = ((4km+2)\Delta + 2m\delta + R)(1+\rho)$.*

Proof. A nonfaulty node may start its fth 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 fth 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

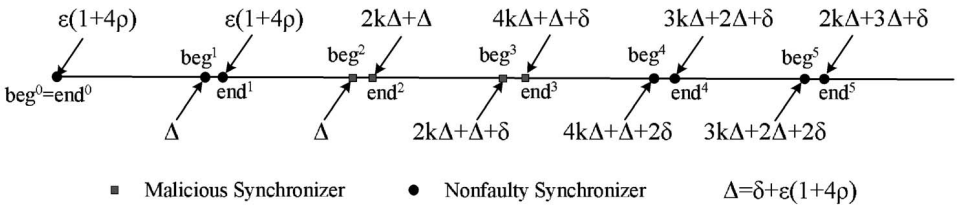

Fig. 2. Changes of clock difference.

have $C_i^f(beg^{f+1}) - C_i^f(beg^f) < (4km+2)\Delta + 2m\delta + R$. By (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 7. *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 (2), over the bounded synchronization interval provided by Lemma 6, 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 2. *When $n > 3m$, Algorithm 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 5 and 7. □

Fig. 2 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 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.

3.4 Discussion

3.4.1 Theoretical versus Average Maximum Clock Differences

Theorem 2 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. Fig. 3 shows the theoretical maximum clock difference and the average maximum clock difference reached in the simulations. We picked n to be 10,

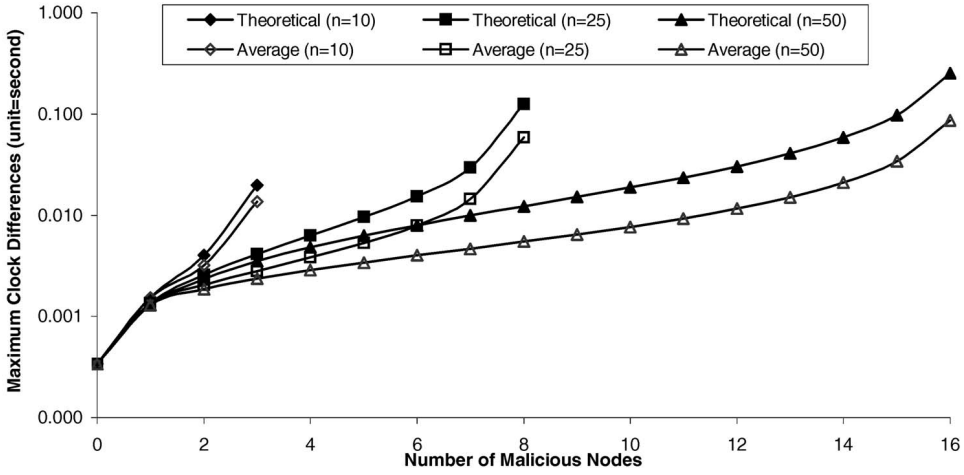

Fig. 3. Theoretical versus average maximum clock differences reached in simulations.

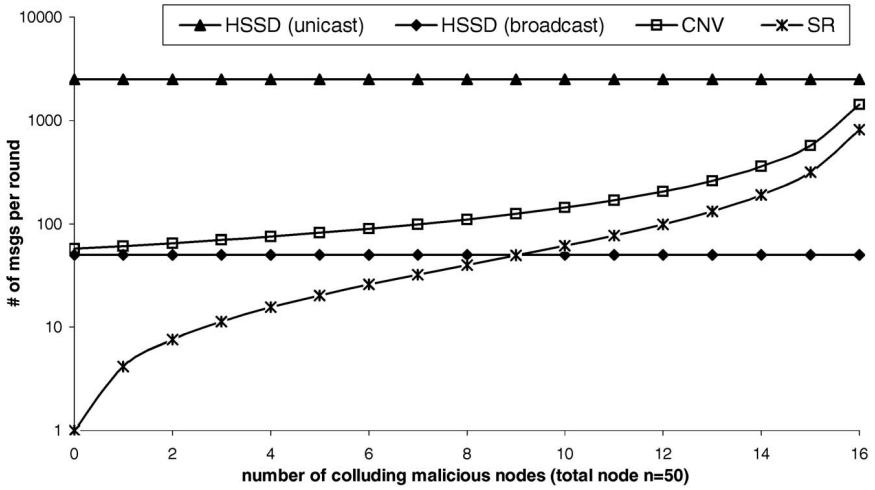

Fig. 4. Communication overhead with the same guarantee of maximum clock difference.

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.

3.4.2 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 ith 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 [13], 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., [41]). 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 [41], the listen time is set to 300 ms, and the sleep time is set to 1 second. According to our simulation result in Fig. 4, our scheme can guarantee that the maximum clock difference is less than 150 ms. Hence, our scheme can be combined with [41] to provide clock synchronization.

3.4.3 Clustering and Global Clock Synchronization

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 clock synchronization in each cluster.

It is simple to deploy a cluster-head in each cluster and achieve a cluster-wise clock synchronization by making other nodes synchronize to the cluster-head. However, if

TABLE 2
Performance Comparison

Algorithm	Degree of Fault-Tolerance	Comm. Overhead (# msgs/round)	Maximum Clock Difference
CNV [19]	$\frac{n-1}{3}$	n^2 (unicast)	$\frac{n}{n-3m}(2\epsilon + 2\rho R)$
COM [19]	$\frac{n-1}{3}$	n^{m+1} (unicast)	$(6m+4)\epsilon + 2\rho R$
CSM [19]	$\frac{n-1}{2}$	n^{m+1} (unicast)	$(m+6)\epsilon + 2\rho R$
$HSSD$ [7]	$n-1$	n^2 (unicast)	$\epsilon + 2\rho R$
SR	$\frac{n-1}{3}$	1 (broadcast)	$(\frac{2mn}{n-3m}+1)\epsilon + \frac{(\frac{2nm}{n-3m}+m+1)}{1-4\rho(\frac{2nm}{n-3m}+m+1)}2\rho R$

the cluster-head is compromised, the whole cluster may not be synchronized correctly. Our scheme provides the fault tolerance property and load balancing through the rotation of synchronizers (cluster-heads).

Our cluster-wise clock synchronization scheme cannot be recursively applied among clusters to provide a global clock synchronization. However, it can potentially be used as a building block of existing global clock synchronization schemes. For example, Olson and Shin developed a fault-tolerant clock synchronization technique [28] that divides a large number of nodes into overlapping synchronization groups and adapts an interactive convergence algorithm [19] within each synchronization group. Alternatively, we may exploit the cluster structure provided by clustering algorithms. One simple solution is to elect a cluster-head in each cluster, and synchronize all the cluster-heads to a trusted external clock source. When the cluster-heads are nonfaulty, each node in a cluster can derive the global clock time through the cluster-head. To increase the degree of fault-tolerance, each cluster may elect more than one cluster-head, and each node can use the median value of all the global clock values derived through these cluster-heads. (Note that, even if all cluster-heads are compromised, the remaining nodes in a cluster can still maintain a synchronized local clock as long as no more than 1/3 of the nodes are compromised.) Clearly, more research is necessary to extend our algorithm to a specific global clock synchronization technique. We consider this as future work.

3.5 Comparison with Previous Techniques

In our proposed algorithm, in each round of clock 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 clock synchronization (when there is no malicious attack). In contrast, almost all of the existing fault-tolerant clock synchronization schemes (e.g., CNV [19], HSSD [7]) 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 [42] 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 [19], HSSD [7]) require digital signatures, which make them undesirable for resource constrained sensor networks. Note that these schemes cannot use this recent authentication technique [42]. 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 2 compares our scheme with existing fault-tolerant clock 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 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 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 3.4).

Fig. 4 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 Fig. 4). 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 Related Work

Clock synchronization problem has been studied for many years. Early techniques (e.g., NTP [25]) are mainly for clock synchronization in wired networks. However, such techniques usually assume there is unlimited computing resource and network bandwidth and, thus, are not suitable for resource constrained sensor networks.

As discussed in the Section 1, several clock synchronization techniques (e.g., [10], [11], [6], [13], [29], [37], [21], [26], [14], [16], [32]) have been proposed for sensor networks recently. Elson et al. developed the Reference Broadcast Synchronization (RBS) scheme for pair-wise as well as multidomain clock synchronization [10], which eliminates the uncertainty of send time and access time from the clock reading error by using a reference broadcast node. Dai and Han improved RBS by reducing the communication overhead in each broadcast domain to two broadcasts [6]. PalChaudhuri et al. proposed a probabilistic clock synchronization based on RBS [29]. A probabilistic clock synchronization in wired network ([1], [5]) has been extended to sensor networks [29] with reduced communication overhead. Generiwal et al. proposed a hierarchical clock synchronization scheme named TPSN for sensor networks [13], assuming clock synchronization messages are timestamped at MAC layer. Sichitiu and Veerarittiphan developed a light-weight scheme to deterministically estimate the bounds on both the relative clock drift and offset between two sensor nodes, which can be used to synchronize their clocks [37]. Li and Rus proposed global clock synchronization techniques only based on local diffusion of clock information [21]. However, all of these techniques assume benign sensor networks, but cannot survive malicious attacks and compromised nodes. The techniques in this paper are developed to address attacks from compromised nodes for cluster-wise clock synchronization.

The proposed techniques are closely related to traditional fault-tolerant clock synchronization in distributed systems, which has undergone substantial research (e.g., [31], [19], [15], [7], [23], [24], [38], [34], [35], [28], [2], [18], [36], [40]). These techniques take either a *software* or a *hardware* approach [31]. Hardware-based techniques require a synchronization circuitry continuously monitor all the clocks [31] and, thus, cannot be used in sensor networks. Software-based techniques can be further classified into convergence-averaging (e.g., CNV [19], LL [23], [24]), convergence-nonaveraging (e.g., HSSD [7], ST [38]), or consistency algorithms (e.g., COM, CSM [19]). Some software-based (or hardware-assisted, hybrid) techniques [30] use hardware to generate timestamps and, thus, can reduce the uncertainty involved in clock synchronization. A common theme of these techniques is to use redundant messages to deal with malicious participants that may behave arbitrarily.

These fault-tolerant 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 [19], HSSD [7]), or a broadcast primitive that requires simultaneous broadcast from multiple nodes, which will result in message collisions in wireless sensor networks. The approach in [28] divides nodes into overlapping clusters and runs one of the previous algorithms (e.g., CNV [19], LL [23], [24]) within each cluster. This reduces the communication overhead, but cannot avoid message collisions in sensor networks. Our techniques in this paper address these problems by exploiting some unique features in cluster-wise clock synchronization in sensor networks and recent results in local broadcast authentication.

5 Conclusion

In this paper, we developed a fault-tolerant clock synchronization scheme for clusters of nodes in sensor networks. Our 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. Compared with the existing fault-tolerant clock synchronization techniques, the proposed scheme can avoid the message collision problem in these approaches, 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 3.1) or a fault-tolerant initial clock synchronization method (e.g., [24]). 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.

Our future work will be focused on the investigation of lightweight, fault-tolerant techniques for global clock synchronization in sensor networks.

REFERENCES

[1] K. Arvind, "Probabilistic Clock Synchronization in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 5, pp. 474-487, 1994.

[2] B. Barak, S. Halevi, A. Herzberg, and D. Naor, "Clock Synchronization with Faults and Recoveries," *Proc. 19th Ann. ACM Symp. Principles of Distributed Computing,* pp. 133-142, 2000.

[3] K. Chakrabarty, S.S. Iyengar, H. Qi, and E. Cho, "Grid Coverage for Surveillance and Target Location in Distributed Sensor Networks," *IEEE Trans. Computers,* vol. 51, pp. 1448-1453, 2002.

[4] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," *IEEE Symp. Research in Security and Privacy,* pp. 197-213, 2003.

[5] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing,* vol. 3, no. 3, pp. 146-158, 1989.

[6] H. Dai and R. Han, "TSYNC: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks," *ACM SIGMOBILE Mobile Computing and Comm. Rev.,* vol. 8, no. 1, pp. 125-139, 2004.

[7] D. Dolev, J.Y. Halpern, B. Simons, and R. Strong, "Dynamic Fault-Tolerant Clock Synchronization," *J. ACM,* vol. 42, no. 1, pp. 143-185, 1995.

[8] J.R. Douceur, "The Sybil Attack," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* Mar. 2002.

[9] W. Du, J. Deng, Y.S. Han, and P. Varshney, "A Pairwise Key Pre-Distribution Scheme for Wireless Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* pp. 42-51, Oct. 2003.

[10] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *ACM SIGOPS Operating Systems Rev.,* vol. 36, pp. 147-163, 2002.

[11] J. Elson and K. Römer, "Wireless Sensor Networks: A New Regime for Time Synchronization," *Proc. First Workshop Hot Topics in Networks (HotNets-I),* Oct. 2002.

[12] A. Galleni and D. Powell, "Consensus and Membership in Synchronous and Asynchronous Distributed Systems," Technical Report 96104, LAAS, Apr. 1996.

[13] S. Ganeriwal, R. Kumar, and M.B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proc. First Int'l Conf. Embedded Networked Sensor Systems (SenSys),* 2003.

[14] J. Greunen and J. Rabaey, "Lightweight Time Synchronization for Sensor Networks," *Proc. Second ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA),* Sept. 2003.

[15] J.Y. Halpern, B.B. Simons, H.R. Strong, and D. Dolev, "Fault-Tolerant Clock Synchronization," *Proc. Third Ann. ACM Symp. Principles of Distributed Computing,* pp. 89-102, 1984.

[16] A. Hu and S.D. Servetto, "Asymptotically Optimal Time Synchronization in Dense Sensor Networks," *Proc. Second ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA),* Sept. 2003.

[17] Y.C. Hu, A. Perrig, and D.B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks In Wireless Ad Hoc Networks," *Proc. INFOCOM 2003 Conf.,* Apr. 2003.

[18] C.M. Krishna, K.G. Shin, and R.W. Butler, "Ensuring Fault Tolerance of Phase-Locked Clocks," *IEEE Trans. Computers,* vol. 34, no. 8, pp. 752-756, Aug. 1985.

[19] L. Lamport and P.M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *J. ACM,* vol. 32, no. 1, pp. 52-78, 1985.

[20] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems (TOPLAS),* vol. 4, no. 3, pp. 382-401, 1982.

[21] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," *Proc. IEEE INFOCOM 2004 Conf.,* Mar. 2004.

[22] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* pp. 52-61, Oct. 2003.

[23] J. Lundelius and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Proc. Third Ann. ACM Symp. Principles of Distributed Computing,* pp. 75-88, 1984.

[24] J. Lundelius-Welch and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Information and Computation,* vol. 77, no. 1, pp. 1-36, 1988.

[25] D.L. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Comm.,* vol. 39, no. 10, pp. 1482-1493, 1991.

[26] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Clock Synchronization for Wireless Local Area Networks," *Proc. 12th Euromicro Conf. Real-Time Systems (Euromicro-RTS 2000),* June 2000.

[27] J. Newsome, R. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Networks: Analysis and Defenses," *Proc. IEEE Int'l Conf. Information Processing in Sensor Networks (IPSN 2004),* Apr. 2004.

[28] A. Olson and K.G. Shin, "Fault-Tolerant Clock Synchronization in Large Multicomputer Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 9, pp. 912-923, 1994.

[29] S. PalChaudhuri, A.K. Saha, and D.B. Johnson, "Adaptive Clock Synchronization in Sensor Networks," *Information Processing in Sensor Networks (ISPN),* Apr. 2004.

[30] P. Ramanathan, D.D. Kandlur, and K.G. Shin, "Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems," *IEEE Trans. Computers,* vol. 39, no. 4, pp. 514-524, 1990.

[31] P. Ramanathan, K.G. Shin, and R.W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *IEEE Computer,* vol. 23, no. 10, pp. 33-42, 1990.

[32] K. Römer, "Time Synchronization in Ad Hoc Networks," *Proc. Second ACM Int'l Symp. Mobile Ad Hoc Networking and Computing,* pp. 173-182, 2001.

[33] K. Römer, P. Blum, and L. Meier, "Time Synchronization and Calibration in Wireless Sensor Networks," *Wireless Sensor Networks,* Ivan Stojmenovic, ed., 2005.

[34] F.B. Schneider, "A Paradigm for Reliable Clock Synchronization," Technical Report TR 86-735, Dept. of Computer Science, Cornell Univ., 1986.

[35] F.B. Schneider, "Understanding Protocols for Byzantine Clock Synchronization," Technical Report TR 87-859, Dept. of Computer Science, Cornell Univ., 1987.

[36] K.G. Shin and P. Ramanathan, "Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults," *IEEE Trans. Computers,* vol. 36, no. 1, pp. 2-12, 1987.

[37] M.L. Sichitiu and C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks," *Proc. IEEE Wireless Comm. and Networking Conf. WCNC03,* 2003.

[38] T.K. Srikanth and S. Toueg, "Optimal Clock Synchronization," *J. ACM,* vol. 34, no. 3, pp. 626-645, 1987.

[39] D. Tian and N.D. Georganas, "A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks," *Proc. First ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA02),* Sept. 2002.

[40] N. Vasanthavada and P.N. Marinos, "Synchronization of Fault-Tolerant Clocks in the Presence of Malicious Failures," *IEEE Trans. Computers,* vol. 37, no. 4, pp. 440-448, 1988.

[41] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient Mac Protocol for Wireless Sensor Networks," *Proc. IEEE INFOCOM 2002 Conf.,* June 2002.

[42] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* pp. 62-72, Oct. 2003.

Kun Sun received the BS degree in 1997 and the ME degree in 2000, both from the Department of Computer and System Science at Nankai University of China. He is currently a fourth-year PhD candidate in the Department of Computer Science at North Carolina State University. His research focuses on wireless network, especially on the security issues in wireless ad-hoc networks and wireless Sensor networks. Before joining NCSU, he had worked as MTS in Bell-labs for one year.

Peng Ning received the ME degree in communication and electronic systems in 1997 and a BS degree in information science in 1994, both from the University of Science and Technology of China. He received the PhD degree in information technology from George Mason University in 2001. He is currently an assistant professor of computer science in the College of Engineering at North Carolina State University. His research interests are mainly in computer and network security. His recent work is mostly in intrusion detection and security in ad hoc and sensor networks. His research has been supported by the US National Science Foundation (NSF), the Army Research Office (ARO), the Advanced Research and Development Activity (ARDA), and the NCSU/Duke Center for Advanced Computing and Communication (CACC). He is a founding member of the NCSU Cyber Defense Laboratory and a member of the NCSU/Duke CACC. He is also a member of the ACM, the ACM SIGSAC, the IEEE, and the IEEE Computer Society.

Cliff Wang graduated from North Carolina State University with a PhD in computer engineering in 1996. He is currently the program director for the Army Research Office's Information Assurance program and manages a large portfolio of advanced information assurance research projects. He is also appointed as an associate faculty member of computer science in the College of Engineering at North Carolina State University. His research interests are in the area of the development of secure protocols and algorithms for wireless sensor networks and wireless mobile ad hoc networks and optimization of sensor network coverage and deployment. He is a member of the IEEE.