© 2011 Maria Berenice Carrasco

OPPORTUNISTIC CLOCK SYNCHRONIZATION FOR AD HOC NETWORKS

BY

MARIA BERENICE CARRASCO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Assistant Professor Sayan Mitra

ABSTRACT

An ad hoc network is a collection of computing nodes communicating over wireless channels without relying on any fixed infrastructure such as servers and towers. Such networks are useful in rescue operations, and in rural and military settings. Clock synchronization is an essential building block for many ad hoc wireless network applications. It provides the participating computing nodes with logical clocks whose differences can be bounded. Several traditional distributed clock synchronization algorithms use strict communication structures such as spanning trees. In such protocols, a node corrects its logical clock when it receives a new time-stamped message from its parent. In this thesis, we present a new clock synchronization protocol that exploits the broadcast medium in wireless networks, allowing nodes to opportunistically correct their logical clocks in order to converge to a reference time provided by a designated root node. Our protocol does not rely on a communication structure and is lightweight due to its low overhead. We also propose a variation of our opportunistic protocol, which further reduces overhead through randomized broadcast techniques. Our simulation-based experimental evaluation of the protocols illustrates that our opportunistic algorithms improve the accuracy of the nodes' logical clocks, when compared to a tree-based protocol. However, we show that the level of improvement is a function of the density of the wireless network. Additionally, the results show that our algorithms can produce around half the overhead, compared to an existing protocol that achieves higher levels of precision.

To my parents and sister, for their love and unconditional support, and to Sukru, for his patience and invaluable help.

ACKNOWLEDGMENTS

First of all, I would like to thank my adviser, Prof. Sayan Mitra, for his invaluable guidance, support and incentive during my years of study at the University of Illinois. I also greatly appreciate the cooperation of Prof. Nitin Vaidya during the development of my research and thesis.

Finally, I am also very grateful to the Comisión Fulbright Ecuador, the institution that sponsored my master's program in the United States. Thanks for the confidence provided and for allowing me to live such a wonderful and enriching experience.

TABLE OF CONTENTS

LIST OF FIGURES vi

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 RELATED WORK 5
2.1 Overview 5
2.2 The Challenges of Clock Synchronization in Wireless Networks 5
2.3 Clock Synchronization Algorithms for Wireless Networks . . . 8
2.4 IEEE 802.11 Standard for Wireless Networks 12

CHAPTER 3 OPPORTUNISTIC CLOCK SYNCHRONIZATION . . 16
3.1 Overview 16
3.2 The System Model 17
3.3 The Algorithms 18

CHAPTER 4 EXPERIMENTAL RESULTS 26
4.1 Overview 26
4.2 Simulation Setup 26
4.3 Opportunistic versus Tree-Based Clock Synchronization 28
4.4 Pure Opportunism versus Randomized Opportunism 32
4.5 Overhead Comparison 35
4.6 WOP and the Extrapolation Problem 39

CHAPTER 5 CONCLUSIONS 40

CHAPTER 6 FUTURE WORK 42

REFERENCES 43

LIST OF FIGURES

4.1 a) Topology 1: 15 nodes and network diameter of 6 hops. b) Topology 2: 20 nodes and network diameter of 4 hops. c) Topology 3: 30 nodes and network diameter of 5 hops. d) Topology 4: 40 nodes and network diameter of 6 hops. e) Topology 5: 60 nodes and network diameter of 6 hops. . . . 27

4.2 a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 1, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms. 30

4.3 a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 2, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms. 30

4.4 a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 3, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms. 30

4.5 a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 4, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms. 31

4.6 a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 5, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms. 31

4.7 Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 1, for different values of δ in milliseconds. 33

4.8 Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 2, for different values of δ in milliseconds. 33

4.9 Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 3, for different values of δ in milliseconds. 34

4.10 Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 4, for different values of δ in milliseconds. 34

4.11 Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 5, for different values of δ in milliseconds. 35
4.12 Illustration of overhead by using TSync on the given topology. (a) Small topology with three broadcasting domains A, B, and C. Node 1 is the root in the network and node 4 is the focus of our analysis. (b) Reception of messages at node 4 for a particular execution. Vertical lines are timelines illustrating the order of events, and oblique lines are messages transmitted from senders to receivers. Since TSync uses a three-message exchange for synchronization, a black line is the last message of this trio, which triggers clock corrections, and a red line is either the first or second message of it. 36
4.13 a) Average error values, $\bar{\varepsilon}$, for Algorithm 4 and Algorithm 2 in topology 2, with different values of δ in milliseconds. . . . 38
4.14 a) Average error values, $\bar{\varepsilon}$, for Algorithm 4 and Algorithm 2 in topology 4, with different values of δ in milliseconds. . . . 38

CHAPTER 1

INTRODUCTION

Many applications for ad hoc networks require synchronized clocks, such as data fusion, target tracking, fault diagnosis and recovery in distributed databases [1]. Each participating node i in a network possesses a *physical clock* HC_i, driven by a hardware oscillator. The frequencies of these oscillators vary over time depending on environmental factors, and therefore, the clock frequencies are also time-varying [2]. It is standard to model the evolution of such clocks as:

$$1 - \rho \leq \frac{d(HC_i)}{dt} \leq 1 + \rho, \tag{1.1}$$

where t is real-time, and ρ is the maximum variation in the frequency of the oscillators specified by the manufacturer. The difference of the values reported by the physical clocks at two nodes is called the *offset* between the nodes' clocks. Because of frequency variations, the offset between the physical clocks can grow unbounded. The clock synchronization problem requires the creation of *logical clocks* LC_i, at each node, derived from the physical clocks and maintained through communication of time-stamped messages, in such a way that the offset between these logical clocks can be guaranteed to be small. Formally, this means that there exists an $\epsilon > 0$ such that for any two nodes i and j in the network, $|LC_i - LC_j| \leq \epsilon$. With this property, higher level applications, like the ones mentioned earlier, can use synchronized logical clocks instead of the physical clocks.

Traditional clock synchronization algorithms designed for wired networks [3, 4, 5] typically depend on reliable communication and do not scale well under the stringent resource constraints which are common in wireless ad hoc networks [2, 6]. Moreover, the quality of synchronization depends crucially on the estimate of the uncertainty in message delays [7], and this uncertainty can be large in wireless networks owing to interference and contention. Thus, the

design of clock synchronization algorithms for ad hoc networks involves trade-offs among communication overhead (message complexity), energy consumption, and accuracy of synchronization. In general, approaches that achieve better synchronization accuracy require more messages to be exchanged in the network, and therefore, require more energy [8]. Energy consumption may be reduced by decreasing the nodes' transmission power and, therefore, the transmission range. This implies a larger number of hops that a message must traverse in order to reach its destination. Even though multi-hop communication moderates the overall power consumption of the network [9], it causes synchronization error to propagate and grow incrementally, affecting the clock synchronization accuracy directly.

In this thesis we present an opportunistic clock synchronization algorithm for wireless ad hoc networks. The key idea behind our protocol is to exploit an important feature of wireless networks, namely, the broadcast medium. Unlike the point-to-point communication links in a wired network, in the wireless setting, a message sent by node A to node B can often be received by a third node C. In the context of synchronization, a time-stamped message intended for node B can aid node C in maintaining the latter's logical clock. It would appear that such opportunistic overhearing is not only possible when communication happens through local broadcasts, but also that it should improve the performance of the synchronization algorithm for free. Typical clock synchronization algorithms for wireless networks [10, 11, 12] still rely on point-to-point communication, and often on a strict tree-like communication structure—and therefore forgo opportunistic updates.

Specifically, our algorithm uses opportunistic gossip-like clock adjustments in a multi-hop wireless network. A synchronization round is started by a designated root node and sequences of logical clock updates cascade through the network. In each round, a non-root node listens for synchronization messages from its neighbors for up to δ seconds, where δ is a parameter of the algorithm called the *window of opportunity (WOP)*. During this window, a node updates its logical clock whenever it receives a synchronization message that complies with a certain metric. Indeed, it may, and in fact often does, update its logical clock multiple times in a given round. Some time after the WOP elapses, the node makes its own time-stamped broadcast, and in this way, the synchronization round propagates through the network. We observed that the performance of our algorithm depends on the size of the

WOP. For instance, as δ increases, the average error of the logical clocks, compared to the reference time, decreases. However, an arbitrarily large δ can introduce additional errors to the logical clocks due to higher levels of contention and collisions. This implies a cost for being opportunistic.

We observed that the density of the network is an important factor that influences the performance of our opportunistic algorithm. In a denser network, the increase of contention and collisions, due to interference, affects negatively the precision of the logical clocks. Therefore, we also propose a variation of the above algorithm, in which the communication overhead is lowered by implementing a randomized broadcast. That is, every node in the network will broadcast a time-stamped message with a certain probability p, which depends on the number of synchronization messages the node has received during the WOP. The idea is to decrease the number of redundant broadcasts in denser areas of the network.

We experimentally evaluate both our algorithms against: **1)** a tree-based algorithm (with no opportunism) proposed in [12], which we refer to as *the baseline*; and **2)** the TSync protocol proposed in [13], which is a level-based algorithm that uses a similar concept to what we call *opportunism*. We implement these protocols in the network simulator NS-2 [14], where we use the IEEE 802.11 standard for wireless communication, and study five topologies under two different traffic scenarios.

Our experimental results indicate that our deterministic opportunistic algorithm performs significanly better than the baseline algorithm for a small window of opportunity (for example, $\delta = 2$ ms). However, the performance is directly affected by the network density. For instance, for our densest topology, our opportunistic algorithm improves the accuracy of the baseline by only about 4%. The TSync protocol performs better than our opportunistic algorithms in terms of accuracy, due to its method of estimating message delays; however, TSync appears to be more expensive in terms of overhead compared to our deterministic opportunistic algorithm (for example, for topology 2, our protocol decreases TSync's overhead by around 47%). Finally, our randomized opportunistic algorithm has even smaller communication overhead compared to the deterministic algorithm (for example, in one of the tested topologies there was a communication overhead reduction of 27%). Somewhat surprisingly, the randomized algorithm performs better than the deterministic algorithm in some situations. The reason is that there

is a smaller number of nodes competing for channel access and, therefore, the uncertainty of the message delays in the network is reduced, improving the accuracy of the synchronization.

CHAPTER 2

RELATED WORK

2.1 Overview

In this chapter we present a summary of relevant clock synchronization protocols designed for wireless networks. We highlight the different characteristics of wireless networks compared to traditional wired networks, so that the challenges of designing distributed algorithms for these types of networks can be better understood. Moreover, we describe the IEEE 802.11 Distributed Coordination Function: the fundamental set of protocols that build up the MAC layer in wireless local area networks (WLANs), whose particularities are part of our experimental environment.

2.2 The Challenges of Clock Synchronization in Wireless Networks

Over the years, many clock synchronization algorithms have been designed for traditional wired networks. Authors in [6] claim that these protocols are not suitable for wireless networks due to the inherent characteristics of this type of network. The following subsections summarize the main differences between wired and wireless networks and how they influence the design of clock synchronization algorithms. The specific protocols referred to in this section will be detailed in Section 2.3.

2.2.1 Limited Energy

Wireless networks, especially ad hoc and sensor networks, are generally deployed without any infrastructure. This means that the network does not

have a centralized administration of the nodes (i.e. base stations, access points), and, moreover, these nodes are not wired to any power source.

Certain operations performed by the nodes in a wireless network, such as using the CPU or transmitting packets, are known for having a considerable cost in terms of power consumption. However, listening to and receiving from the network require significant energy as well, compared to the overall system budget [8].

In general, clock synchronization algorithms are to be performed periodically, as mentioned in Chapter 1. Since the nodes in a wireless network have a limited lifetime, clock synchronization algorithms should be designed such that their purpose is achieved while preserving energy to utilize the nodes in an efficient fashion. For example, traditional clock synchronization protocols such as NTP [3] use an external standard like GPS (Global Positioning System) or UTC (Universal Time Coordinated) in order to synchronize the network to an accurate time source. However, the use of GPS poses a high demand for energy and, as a result, it is mainly avoided in protocols for wireless networks.

Reduction of energy consumption is achieved by choosing to transmit over multiple short distances instead of a single long path. This translates into either a lower transmit power or a higher data transmission speed over a given distance. Either one will decrease the total end-to-end energy needed to transmit a packet of data. This implies that in large wireless networks, data is transmitted in sequences or hops, instead of a single long path from the sender to the receiver. However, as will be addressed in the following subsection, multi-hop networks increase the end-to-end delay in the network.

2.2.2 End-to-End Transmission Delay

In traditional wired networks, a single constant end-to-end transmission delay and variance bound are considered for all the messages in the system because any node can send a message directly to another node at any point in time [9]. Therefore, data is transmitted along a single hop. On the other hand, wireless networks might involve many hops, depending on the size of the network and the transmission power used by the nodes, which determines their range of coverage.

Along with the multi-hop characteristic of wireless networks, additional features make it impractical to assume a single end-to-end transmission delay bound between a sender and a receiver in the network:

- The wireless transmission over a shared medium implies packet collisions and contention [15], as will be further explained later in this chapter.
- The state of a wireless link between any two nodes is non-deterministic because it varies over time. Thus, the end-to-end delay becomes time-dependent as well.
- The end-to-end delay is not symmetric, meaning that it might be different in the same wireless link, depending on the direction of the message (i.e. from node A to node B or from node B to node A).

All the characteristics explained above result in the inability to correctly estimate the end-to-end delay for message transmission. Therefore, this non-determinism implies an unpredictable variation in transmission times which is often referred to as *uncertainty*. Authors in [8] emphasize that the most important source of error in clock synchronization is, indeed, the uncertainty of the message delay because it makes it difficult for a receiver to estimate the time at which a message was sent and vice versa. In general, the time involved in sending a message from a sender to a receiver is the result of the following four factors, all of which can vary non-deterministically:

1. Send delay: the time it takes for the sender to build the message and transmit it to the network interface.
2. Access delay: the time the sender spends waiting to access the wireless channel in order to transmit the message. This delay is highly influenced by the number of nodes located in the sender's transmission range.
3. Propagation delay: the time taken for the message to reach the receiver, once it has left the sender.
4. Receive delay: the time spent by the receiver to process the message.

Due to these different types of delays involved, a clock synchronization algorithm can reduce its introduced error by circumventing one or some of these delays. For example, the RBS protocol [16] directly removes two of the largest sources of non-determinism involved in message transmission, namely the send time and the access time. Thus, this protocol can provide a high degree of synchronization accuracy in wireless networks.

2.2.3 Dynamic Topology

In traditional wired networks, despite temporary failures, the topology remains relatively static. Thus, for clock synchronization protocols that depend on hierarchical structures (i.e. spanning trees), the nodes meant to function as reference sources of time are generally manually configured if the underlying network is wired. This is the case of NTP [3].

On the other hand, wireless networks are dynamic whether the nodes are mobile or not. In fact, despite nodes can be deployed in specific locations, the wireless medium is unshielded to external interference (i.e. noise, obstacles, simultaneous transmissions), which may lead to a high percentage of message loss and, therefore, intermittent connectivity [15]. As a result, an additional challenge is added to the design of distributed systems for wireless networks: self-configuration. Some clock synchronization protocols for wireless networks, such as TSync [13], assume that each node is aware of its neighboring nodes. However, due to the dynamic nature of the network, the use of suitable neighbor discovery or leader election protocols is necessary to achieve synchronization.

2.3 Clock Synchronization Algorithms for Wireless Networks

Several clock synchronization algorithms have been presented in the literature for wireless ad hoc networks. While some of these protocols aim to achieve the best possible precision of synchronization, others minimize the resource requirements of the network. We refer the reader to [8] for a survey and to [2] for a summary of the key design principles that authors have identified for developing clock synchronization algorithms. Depending on the

application requirements, clock synchronization algorithms balance among accuracy, energy efficiency and overhead. Highly accurate synchronization methods increase the consumption of energy and usually have high message and computational complexity. On the other hand, energy-efficient protocols sacrifice precision to extend battery life. The following subsections describe some of the most important clock synchronization protocols that we consider relevant for our study.

2.3.1 The Reference-Broadcast Synchronization

Reference-Broadcast Synchronization (RBS) [16], was the first algorithm that exploited the broadcast nature of the wireless channel by implementing receiver-to-receiver handshaking. This approach requires a reference node to transmit a beacon; then, the receivers synchronize with each other by exchanging the time-stamp at which they received the reference's beacon, and by computing their offset according to the difference in the reception times. This results in a reduction of the message delay variance, since the message is time-stamped using only the receiver's clock.

Furthermore, in order to increase accuracy, the reference node broadcasts m beacons per synchronization round, so that each receiver considers the average of the phase offsets with other nodes in the network. The authors in [16] account for clock skew by performing least-squares linear regression instead of averaging the phase offsets; the slope and intercept of the line are exploited to recover the frequency and the phase of the nodes' clocks with respect to the remote node [16]. Finally, the authors also extend their work for multi-hop synchronization with some accuracy lost.

In RBS, instead of adjusting the logical clocks, each node builds a table of parameters that relate its clock to the logical clock of every other node in the network. For instance, when node A receives a timestamp from B, it must look at its table to find an entry for node B and determine how to interpret its timestamp.

Despite the improvement in clock precision, this algorithm requires large communication overhead[1] compared to traditional sender-to-receiver handshaking [2], since the number of necessary broadcasts increases dramatically

[1]Overhead has a direct implication on energy consumption levels [6].

due to pairwise verification.

2.3.2 The Flooding Time Synchronization Protocol

The Flooding Time Synchronization Protocol (FTSP) [17, 18] is based on RBS, and utilizes broadcast messages and linear regression. Moreover, it benefits from time-stamping at the MAC layer of the Open System Interconnection (OSI) model, intending to increase accuracy by eliminating the non-determinism caused by source processing (send delay), queuing delay, and channel access delay.

In FTSP, nodes synchronize their clocks according to a reference node. Therefore, this protocol aims to cope with node failures and topology changes by performing leader election [19] dynamically for the reference node. However, one drawback of FTSP is that the method for choosing the leader is based on the assumption that the network is synchronous. This is not the case especially in wireless networks, as mentioned in Section 2.2.

Additionally, in order to account for multi-hop networks, the nodes form an ad hoc hierarchical structure, based on node identifiers, rooted at a reference node. This structure is used to transfer the reference's time to all nodes so they can adjust their logical clocks and converge to the reference time. Thus, the nodes that are not in the single-hop range of the root node synchronize themselves with those that are closer to it. The Network Time Protocol (NTP) [3] is one of the oldest protocols using hierarchical structures. Although NTP is not suitable for wireless networks [6], its layer-based approach has inspired many clock synchronization algorithms for ad hoc wireless networks such as FTSP, whose decoupling from round-trip time (RTT) delay computation makes it a better approach for wireless networks.

2.3.3 Hierarchy-Based Clock Synchronization Protocols

The Simple Network Time Protocol (SNTP) [10] and the Timing-sync Protocol for Sensor Networks (TPSN) [11] are based on hierarchical structures since they construct a spanning tree from a reference node (i.e. based on the distance of each node to the reference) that provides the time for the network. The idea is that every node synchronizes with its neighbor in the

lower level, resulting in a completely synchronized network. The construction of the hierarchical structure is solely based on simple broadcast messages originated from the reference node.

SNTP and TPSN, contrary to RBS, exploit the standard sender-to-receiver handshaking, where a reference node propagates its clock value which is used by the receivers to synchronize themselves with the reference. With the handshaking approach, delays and offsets are computed through an estimation of the RTT. SNTP introduces the two-way message exchange process [10] for such an estimation, and TPSN uses this approach as well. Moreover, in TPSN, by time-stamping at the MAC layer, the error caused by uncertainty at the sender side is alleviated.

Emphasis on energy efficiency has led to approaches that aim to reduce the overall overhead such as the Chaining Clock Synchronization (CCS) protocol [20]. CCS takes advantage of the overhearing technique, feasible in wireless networks. For instance, CCS chooses a subset of nodes in the network to form a spanning tree which is called *skeleton* tree. Then a protocol similar to TPSN is used on top of this skeleton tree. Nodes that do not belong to the tree, *passive nodes*, will synchronize according to overheard messages. Additionaly, the two-way message exchange is achieved by a single message transmission per node, meaning that when a node requests a sync pulse, it replies to its children at the same time ; this requires message propagation from the leaf nodes to the root. CCS considerably reduces energy consumption but does not perform better than TPSN in a single-hop domain; however, it introduces a skew propagation technique that reduces cummulative error in a multi-hop network.

2.3.4 The TSync Protocol

The TSync protocol [13], proposed by Dai and Han, is based on RBS when using the broadcast channel, and also performs two-way message exchange as in SNTP. TSync exploits multiple channel radios for reducing message delays uncertainties, and incorporates two components: **1)** a periodic and **2)** an on-demand clock synchronization process, used according to application needs.

The Hierarchy Referencing Time Synchronization (HRTS) protocol is the

periodic algorithm of TSync, meaning that it is performed every Δ seconds in order to account for clock skew, as mentioned in Chapter 1. In HRTS, the source node first broadcasts a beacon message on a shared *control* channel. An adjacent node, specified by the source, sends a reply on a unique *clock* channel to which the source is also tuned. This reply completes the two-way message exchange used by the source node to calculate the offset; then, this value is sent into a broadcast message to all nodes again, allowing them to synchronize. This process is repeated on different layers to cover the entire network. HRTS aims to decrease the number of broadcast messages incurred by RBS-based algorithms, and allows nodes to correct their logical clocks multiple times per synchronization round. Due to this latter characteristic, we will compare our work with TSync-HRTS in Chapter 4.

The on-demand component of TSync is called the Individual Time Request (ITR) Protocol. By using this protocol, any node is able to synchronize the portion of the network that is needed for its purpose. This mechanism is highly advantageous in energy-constrained networks, and in networks in which only a small portion of the network needs to be synchronized temporarily.

2.4 IEEE 802.11 Standard for Wireless Networks

Since practical real-time communication applications should be based on the IEEE 802.11 standard for wireless communication [21], we implement our algorithms in the NS-2 simulator under realistic characteristics and constraints of IEEE 802.11. Briefly describing this standard is essential in order to understand the results of our simulations and our conclusions.

2.4.1 IEEE 802.11 Distributed Coordination Function

The IEEE 802.11 protocol specification covers the medium access control (MAC) and physical layers of the OSI model. Beyond the standard functionality of coordinating the access to the shared wireless channel, the 802.11 MAC layer performs other functions that are typically related to upper layer protocols, such as fragmentation, packet retransmissions, and acknowledgements.

The MAC layer defines two different access protocols: centralized and distributed. The latter is referred as the distributed coordination function (DCF), which suits multi-hop ad hoc and hybrid networks [15]. This is the wireless MAC protocol used in our network simulator, NS-2, and, therefore, we will focus on its functionality.

Carrier Sensing

DCF is a carrier sense multiple access/collision avoidance (CSMA/CA) protocol, based on the "listen-before-you-talk" premise. For instance, every node desiring to transmit senses the wireless channel; if the channel is busy (i.e. some other node is transmitting) then the node will defer its transmission to a later time; if the channed is sensed as idle then the node is allowed to transmit. The way to determine that the channel is busy is by sensing if the energy on it is above a threshold called the carrier sense (CS) threshold.

This kind of protocol is very effective when the medium is not heavily loaded, since it allows nodes to transmit with minimum delay, but there is always a chance of nodes transmitting at the same time (producing collisions), caused by the fact that the nodes sensed the medium as idle and decided to transmit at once.

Back-off Interval

With the basic scheme described above, consecutive collisions might occur due to nodes sensing the channel as idle at the same time again, right before a failed attempt. In order to reduce the probability of collisions resulting from contention between different nodes willing to access the channel, DCF uses a back-off interval.

Whenever a node wants to transmit, and the channel is sensed as busy, it will choose a back-off interval N (random number) in the range $[0, cw]$, where cw is called the *contention window*. Then, the node sets up a counter and it waits to transmit at the Nth valid opportunity; that is, every time the node senses the channel as idle it counts down the back-off interval, and only when the counter hits zero is the node allowed to transmit.

The time spent counting down back-off intervals is considered as MAC overhead [15]. Choosing a large cw leads to large back-off intervals and can

result in larger overhead. On the other hand, choosing a small *cw* leads to a larger number of collisions (i.e. two nodes count down to 0 simultaneously).

Since the number of nodes attempting to transmit simultaneously may change with time, IEEE 802.11 DCF defines a mechanism called the Exponential Back-off Algorithm, where *cw* is chosen dynamically depending on collision occurrence. Therefore, when a node's message collides, the node increases the contention window exponentially, that is, *cw* is doubled (up to an upper bound). Only when a node completes a data transfer successfully does it restore *cw* to the default value.

Reliability

Wireless links are prone to errors, so DCF implements some mechanisms to reduce the packet loss rate experienced by upper layers. In fact, a simple solution that aims to achieve transmission reliability is a retransmission scheme, where an acknowledgement (ACK) is expected by the sender whenever it transmits a unicast packet. The reception of the ACK will indicate to the sender that no collision occurred. If the sender does not receive the ACK, then it will retransmit the fragment until it gets acknowledged or thrown away after a certain number of retransmissions.

In wireless networks, collisions may occur due to a common problem called hidden terminal. This problem is triggered by two nodes attempting to transmit to a third common destination at the same time. These two senders do not sense each other's transmissions using the carrier sense mechanism since they are not in each other's range of coverage (they are "hidden" to each other). Therefore, both nodes send their packets to the destination, where they collide. In order to reduce the probability of two nodes colliding due to the hidden terminal problem, DCF defines an additional mechanism called virtual carrier sensing.

A node willing to transmit a packet will first send a control packet called request-to-send (RTS), which will include the source, destination, and the duration of the following transaction (i.e. the packet and the respective ACK); the destination node will respond (if the channel is idle) with an acknowledgment control packet called clear-to-send (CTS), which will include the same duration information. If the CTS is not received, the sender will retransmit the RTS.

All nodes receiving the RTS and/or the CTS, will set their virtual carrier sense counter, called Network Allocation Vector (NAV), for the given duration, and will use this information together with the physical carrier sense when sensing the channel.

Virtual Carrier Sensing reduces the probability of a collision in the receiver area by a node that is "hidden" from the transmitter, to the short duration of the RTS transmission, because the node will hear the CTS and "reserve" the medium as busy until the end of the transaction. The duration information on the RTS also protects the transmitter area from collisions during the ACK transmission, by nodes that are out of range from the acknowledging node.

In IEEE 802.11 DCF, virtual carrier sensing is optional while physical carrier sensing is mandatory. Since virtual carrier sensing incurs extra overhead, we disabled it in our simulations in NS-2. The reason is that we do not want to ensure that a packet arrives to its destination since clock synchronization algorithms are periodically executed.

CHAPTER 3

OPPORTUNISTIC CLOCK SYNCHRONIZATION

3.1 Overview

In this chapter, we propose opportunistic clock synchronization algorithms in which nodes simply broadcast synchronization messages, which are further propagated in the network in a gossip-like fashion. Our algorithm does not depend on a particular hierarchical communication structure, such as a tree, and uses opportunistic communication to adjust the logical clocks. Informally, in a tree-based protocol, nodes listen to their parents and transmit to their children. However, in a wireless network, a node could potentially benefit from a synchronization packet that comes not from its parent but from another node that is equally or better positioned in the network with respect to the reference node (i.e. fewer or equal number of hops away from the reference). This intuition provides the basis for our algorithm: During a synchronization round, a node has the opportunity to correct its clock from more than one synchronization message.

Uncertainty in message delays determines the worst-case inaccuracy in clock synchronization [7]. Authors of clock synchronization algorithms aim to reduce the variance of message delays by incorporating different methods in their protocols, but not even the standard two-way message exchange technique can predict round-trip delays completely accurately. Therefore, our protocols do not measure message delays, but instead attempt to minimize uncertainty by time-stamping at the MAC-layer. Our delay estimation considers the propagation delay [22] only (refer to Chapter 2).

In order to compare the performance of our opportunistic algorithms, we introduce two existing clock synchronization algorithms. One of them is a tree-based protocol, which relies on known delays. The second protocol estimates end-to-end message delays and is based on levels. This chapter

details these protocols and provides pseudo-code for all of them.

3.2 The System Model

For our algorithms, we consider the following system model and assumptions, based on that in [12].

We consider a set of n nodes $V = \{v_1, v_2, ..., v_n\}$ in a plane, where each node v_i has static coordinates represented by the Euclidean vector $\overline{x}_i \in \mathbb{R}^2$. One of these nodes is designated as a reference node called the *root*. The distance between any two nodes v_i and v_j is the Euclidean distance, denoted as $dist(\overline{x}_i, \overline{x}_j)$. The propagation delay of a message that is transmitted from v_i to v_j is defined as $\tau_{ij} = dist(\overline{x}_i, \overline{x}_j)/c$, where c is the speed of light. Every node v_i uses the same transmission power P, which determines the maximum distance at which a recipient can reliably receive the message; the power required for broadcasting to distance d is given by the Two-Ray Ground Propagation Model [23].

Each node v_i has a physical and a logical clock defined as follows:

- The physical clock of v_i is modeled as $HC_i(t)$, which evolves according to Equation 1.1. The rate at which the physical clock runs is called σ_i.
- The logical clock of v_i is defined as $LC_i(t) = HC_i(t) + adj_i(t)$, where $adj_i(t)$ is the adjustment value computed by the algorithm at time t for correcting the hardware clock $HC_i(t)$.

Due to the difference in the clock skews, our algorithms will be periodically executed every Δ seconds, where each execution is called a *round*. Only the root node initiates a new round. Finally, we define the clock synchronization *error* of node v_i as the offset between its logical clock and the root's clock, and it is denoted as $\varepsilon_i(t) = HC_{root}(t) - LC_i(t)$. We assume that the root reads its clock value directly from its hardware clock since it provides a reference time for the entire network.

3.3 The Algorithms

3.3.1 Baseline Clock Synchronization Algorithm

As our first comparison baseline we implement a variation of the algorithm proposed by Attiya *et al.* in [12]. Their protocol relies on the construction of a shallow spanning forest. That is, different spanning trees are rooted from different reference nodes. These trees minimize the cummulative message delay uncertainties along the paths, assuming that the mean message delays per link and their uncertainties are known. The authors also assume that these uncertainties are symmetric on the links; that is, for a pair of nodes v_i and v_j, the message delays from v_i to v_j and from v_j to v_i are the same. Once the spanning trees are created, the reference nodes propagate their clock values; the receivers adjust their logical clocks by adding the mean message delay to the sender and the received time-stamp; then they transmit their corrected logical clock values to their children. This protocol assumes that clocks do not have skews at all but only offsets; thus, clock synchronization is performed only once.

In our variation, the algorithm uses one reference node, *root*. Since we assume that clocks skew from the real-time rate, we require the algorithm to be executed periodically. Moreover, our protocol constructs the spanning tree by minimizing the cummulative propagation delay along the paths. In fact, our algorithm does not depend on message delay uncertainties, as [12], since synchronization messages are time-stamped at the MAC-layer. Algorithm 1 describes our baseline clock synchronization protocol after the shortest-path tree has been constructed by using Dijkstra's algorithm [24]. Each node v_i has a local variable $parent_i$, assigned during the tree construction phase, which indicates the node that v_i should listen to in the synchronization phase. Note that every node, v_i, is aware of its position, $\overline{x}_i$, in the Euclidean plane for computing the distance to any other node.

In Algorithm 1 only the root node initiates the synchronization process by broadcasting a *sync* packet. The other broadcasts are triggered whenever a node v_i receives a *sync* packet from its parent. In that case, the distance from v_i to its parent, v_j, is computed through the function $dist(\overline{x_i}, \overline{x_j})$, and the propagation delay, τ_{ij}, is calculated. Node v_i's logical clock will simply get the value of the received time-stamp T plus τ_{ij}. Then v_i immediately

Algorithm 1 Baseline Clock Synchronization Algorithm executed by v_i every Δ time

```
1: Every Δ time:
2: if v_i = root then
3:    broadcast ⟨sync, HC_i, i, x̄_i⟩
4: end if
5:
6: Upon receiving ⟨sync, T, j, x̄_j⟩
7: if parent_i = j then
8:    τ_ij ← dist(x̄_i, x̄_j)/c
9:    adj_i ← (T + τ_ij) − HC_i
10:   LC_i ← HC_i + adj_i
11:   broadcast ⟨sync, LC_i, i, x̄_i⟩
12: else
13:   Discard packet
14: end if
```

broadcasts this value to its neighbors. This process is periodically executed every Δ seconds, where Δ is a parameter of the protocol chosen according to the maximum tolerance for synchronization error and the energy constraints for the system.

Attiya *et al.* prove their synchronization to be optimal, given their system model. In our baseline algorithm, by minimizing the cummulative propagation delay in the tree paths, we are minimizing the diameter of the network and, therefore, the clock synchronization error ε. This informally justifies why this baseline approach is the best we can do given our system model and assumptions.

3.3.2 Opportunistic Clock Synchronization Algorithm

We propose an opportunistic clock synchronization algorithm where no hierarchical structure is needed and synchronization is achieved by designating one of the n nodes as the *root*. In a synchronization round k, node v_i has the opportunity to correct its clock as many times as *sync* messages are received during a time interval of δ seconds which we call the *window of opportunity* (WOP).

Not all the messages heard by v_i will be good for correcting LC_i. For instance, a message coming along a longer path than the first message received

by v_i might carry more error, due to the number of hops, affecting the accuracy of LC_i. In order to address this situation, a metric is necessary for v_i to decide on its reaction towards a particular *sync* message received. Therefore, for each node v_i, we define $metric_i$ as the number of hops along which the first *sync* message has being propagated from the root. Then, we say that a synchronization message is *useful* for v_i if it came from a path with number of hops smaller or equal to $metric_i$.

Algorithm 2 presents our oportunistic approach, where the definition of the following additional variables is necessary:

- δ-timer: The WOP for v_i, which once initialized, elapses for δ seconds according to HC_i's rate.
- seq_num_i*:* A sequence number included in each synchronization message sent by v_i.
- *hops:* Part of the synchronization message received by v_i; identifies the number of hops along which the message has been propagated from the root.

In Algorithm 2, the root will initiate a new synchronization round k every Δ seconds, as in Algorithm 1. In order to ensure that clock corrections are based on fresh information, a sequence number is included in every packet. Note that only the root increases this sequence number as a new round begins. As soon as node v_i hears a *sync* broadcast message, the sequence number is evaluated: the reception of a number larger than v_i's own sequence number triggers the initialization of a timer of δ seconds, WOP, during which v_i will accept packets for potentially correcting its logical clock. Only a packet that has propagated through a number of hops less than or equal to v_i's metric, and for which the sequence number corresponds to the one that started the timer, will cause an update of the logical clock LC_i. When the WOP expires, v_i broadcasts its own logical clock value and no longer accepts messages (i.e. v_i discards them) until the next round.

Algorithm 2 Opportunistic Clock Synchronization Algorithm executed by v_i every Δ time

Every Δ time:
Initialization
$metric_i \leftarrow -1$

if $v_i = root$ **then**
  $seq_num_i \leftarrow seq_num_i + 1$
  broadcast $\langle sync, seq_num_i, HC_i, i, \overline{x}_i, 0\rangle$
end if

Upon receiving $\langle sync, seq_num_j, T, j, \overline{x}_j, hops\rangle$

if $seq_num_j > seq_num_i$ **then**
  Initialize δ-timer
  $seq_num_i \leftarrow seq_num_j$
  $metric_i \leftarrow hops + 1$
end if
if δ-timer is active **then**
  if $seq_num_j = seq_num_i$ **and** $hops + 1 \leq metric_i$ **then**
    $\tau_{ij} \leftarrow \frac{dist(\overline{x}_i, \overline{x}_j)}{c}$
    $adj_i \leftarrow (T + \tau_{ij}) - HC_i$
    $LC_i \leftarrow HC_i + adj_i$
  else
    Discard packet
  end if
end if

Upon expiration of δ-timer
broadcast $\langle sync, seq_num_i, LC_i, i, \overline{x}_i, metric_i\rangle$

3.3.3 Randomized Opportunistic Clock Synchronization Algorithm

Algorithm 2 is based on FTSP [18] since the root's reference time is propagated in the network by broadcasting messages. Note that the inclusion of the WOP does not influence the number of broadcasts per node. For instance, although a node might update more than once per round, it will broadcast its own logical clock value at most once. To further decrease the number of synchronization broadcasts, we propose a variation of Algorithm 2, where we include randomized broadcasts as in [25]. That is, nodes do not necessarily broadcast a *sync* message at the end of the WOP. These nodes broadcast with a certain probability p, instead. Therefore, the probability of node v_i to broadcast LC_i in round k depends on the number of synchronization messages received during that particular round, even though some of those messages were not considered useful for v_i. In other words, the more messages heard by v_i in k, the more nodes in its vecinity; thus, v_i's broadcasting probability decreases as those other nodes might be broadcasting as well. Since the inclusion of randomized broadcasts represents an optimization of our opportunistic approach, Algorithm 3 indicates the lines that are required to be added and replaced in Algorithm 2.

Algorithm 3 Required changes in Algorithm 2 for implementing Randomized Opportunistic Clock Synchronization

Line 4, add:
$collected_i \leftarrow 0$

Line 11, add:
$collected_i \leftarrow collected_i + 1$

Line 28, replace by:
$p_i \leftarrow \frac{1}{collected_i}$
if $random(0,1) \leq p_i$ **then**
  broadcast $\langle sync, seq_num, LC_i, i, \overline{x}_i, metric_i \rangle$
else
  Do not broadcast LC_i
end if

In Algorithm 3, $collected_i$ represents the number of messages that node v_i heard during round k, and the probability of v_i for broadcasting LC_i when the WOP expires is inversely proportional to $collected_i$. For example, if v_i

received only 1 message, the implication is that it had only one neighbor and, therefore, it needs to broadcast with $p_i = 1$. On the other hand, if v_i received 5 messages, it will broadcast with lower probability, $p_i = \frac{1}{5}$, since the implication is that it has at least 5 neighbors that could take its place for broadcasting the synchronization message.

3.3.4 The TSync Protocol

As our second comparison baseline we implement the TSync protocol, proposed by Dai and Han in [13]. Specifically, we address the TSync's periodic clock synchronization process, HRTS, as indicated in Chapter 2. The HRTS protocol does not rely on a strict tree; instead, a hierarchical structure is constructed around a reference node by using the notion of *levels*, which indicate how far away a particular node is from the reference. Moreover, HRTS uses the two-way message exchange technique in order to compute the offset between two logical clocks and adjust them. This message exchange occurs between a source node (i.e. initially the reference node) and some neighboring node specified by it. The rest of the neighbors update their clocks by using the offset computed and broadcasted by the source, and by computing an additional offset themselves.

In our implementation, as in [13], we assume that each node v_i knows about its neighbors, and one of them is assigned as v_i's chosen neighboring node, $chosen_i$, with which the two-way message exchange is performed. Algorithm 4 presents the TSync protocol for a single-channel ad hoc network.

In Algorithm 4, the reference node (i.e. root) initiates the synchronization process by broadcasting a sync_begin message at time t_1. This message specifies $chosen_i$, which is the node that will reply back to the sender in order to complete the two-way message exchange. When the neighboring nodes receive the sync_begin message, they store a time-stamp $t_2{}'$. Additionally, a sequence number is evaluated in order to determine whether a new synchronization round has begun; in that case, v_i will assign its level regarding to the sender's level. The node $chosen_i$ replies back to the sender at time t_3 with a message that includes this time-stamp and the time t_2 at which the sync_begin packet was received. When receiving a reply message at time t_4, the reference node has all the time-stamps required to compute the offset

Algorithm 4 TSync Algorithm executed by v_i every Δ time

Every Δ time:
Initialization
$level_i \leftarrow 0$

if $v_i = root$ **then**
  $seq_num_i \leftarrow seq_num_i + 1$
  $t_1 \leftarrow HC_i$
  broadcast $\langle sync_begin, seq_num_i, chosen_i, 0\rangle$
end if

Upon receiving $\langle sync_begin, seq_num_j, chosen_j, level_j\rangle$
$t_2{}' \leftarrow LC_i$
if $seq_num_j > seq_num_i$ **then**
  $seq_num_i \leftarrow seq_num_j$
  $level_i \leftarrow level_j + 1$
end if
if $level_j < level_i$ **and** $seq_num_j = seq_num_i$ **then**
  if $v_i = chosen_j$ **then**
    $t_2 \leftarrow t_2{}'$
    $t_3 \leftarrow LC_i$
    unicast to v_j $\langle reply, seq_num_i, t_2, t_3\rangle$
  end if
else
  $t_2 \leftarrow null$
end if

Upon receiving $\langle reply, seq_num_j, t_2, t_3\rangle$
$t_4 \leftarrow LC_i$
if $seq_num_j = seq_num_i$ **then**
  $\theta_i \leftarrow \frac{(t_2-t_1)-(t_4-t_3)}{2}$
  broadcast $\langle sync, seq_num_i, t_2, \theta_i\rangle$
else
  Discard packet
end if

Upon receiving $\langle sync, seq_num_j, t_2, \theta_j\rangle$
if $t_2{}' \neq null$ **and** $seq_num_j = seq_num_i$ **then**
  $\theta_i{}' \leftarrow t_2 - t_2{}'$
  $LC_i \leftarrow LC_i + \theta_i{}' + \theta_j$
  $t_1 \leftarrow LC_i$
  broadcast $\langle sync_begin, seq_num_i, chosen_i, level_i\rangle$
else
  Discard packet
end if

θ_i, with respect to $chosen_i$; this concludes the two-way message exchange. Finally, the reference node broadcasts a sync message that includes t_2 and θ_i. When receiving this sync message, since the rest of the neighbors had already registered a receiving time t_2', they compute their own offset θ_i' with respect to the reference's chosen node by using t_2. This local offset θ_i' plus the one computed by the reference node are used to correct the node's logical clock LC_i. Once the nodes have adjusted their logical clocks, they now become reference nodes and repeat the procedure in order to allow downstream nodes to update their clocks as well.

Note that $chosen_i$ sends a reply message only when the sender's level is less than its own level. This allows a node v_i to potentiallly correct its logical clock more than once per round.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Overview

In this chapter, we evaluate the performance of our opportunistic clock synchronization algorithms on the network simulator NS-2 [14]. Since practical real-time communication applications should be based on the IEEE 802.11 standard for wireless communication [21], we implement all the algorithms presented in Chapter 3 in the NS-2 simulator under realistic characteristics and constraints of IEEE 802.11.

First, we describe our simulation setup and the terminology required for the following sections. Then, we present our experimental results against the protocols chosen for comparison. The results evidence the advantages of *opportunism*, upon the baseline algorithm, in improving accuracy while reducing overhead, which is further decreased by our randomized broadcast implementation. However, the TSync protocol provides more accurate synchronization than our proposed algorithms, despite the costly overhead involved.

4.2 Simulation Setup

We used a topology generator to place nodes randomly in a 250 m $\times$ 250 m grid. Figure 4.1 shows the various topologies analyzed. Moreover, in this thesis work we used some of these topologies under two different traffic scenarios: **1)** No traffic, and **2)** Constant Bit Rate (CBR) combined with FTP traffic. Every node used the two-ray ground propagation model and a transmission power such that the maximum radius of coverage is 30 m. With IEEE 802.11 as the MAC layer, we did not use reliable transmissions,

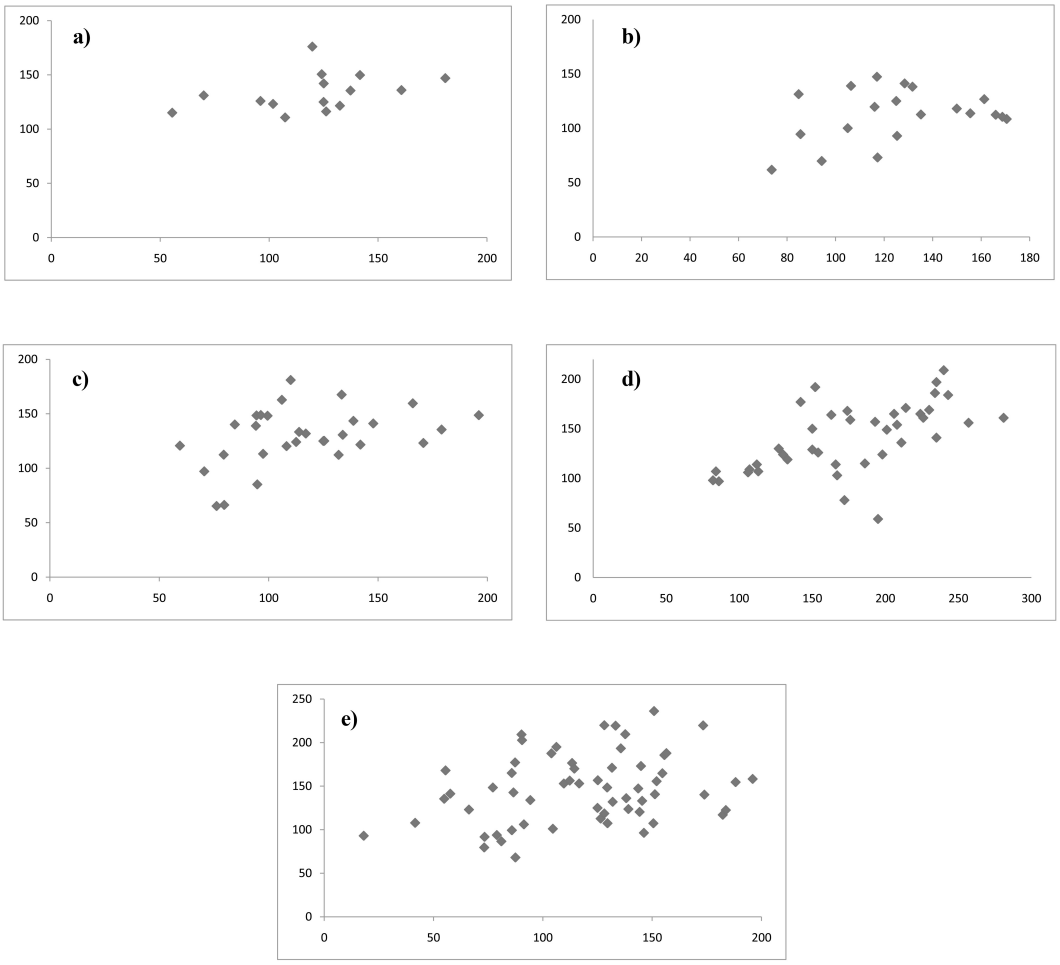


Figure 4.1: a) Topology 1: 15 nodes and network diameter of 6 hops. b) Topology 2: 20 nodes and network diameter of 4 hops. c) Topology 3: 30 nodes and network diameter of 5 hops. d) Topology 4: 40 nodes and network diameter of 6 hops. e) Topology 5: 60 nodes and network diameter of 6 hops.

and virtual carrier sensing (RTS/CTS) was disabled as well as the routing protocol.

As for the nodes' clocks, in this evaluation we consider $\rho = 100$ ppm (parts per million, i.e. a clock can lose up to 100 μs every second), so that the skew for every node was randomly selected from the range $[0.9999, 1.0001]$. Without loss of generality, we assume that the root's hardware clock is perfectly synchronized with real-time. Finally, as in [13], the root initiates a new synchronization round every $\Delta = 10$ s.

For measuring the nodes' logical clock errors with respect to the root's clock, or real-time in this case, we introduce the notion of *snapshots*. A

snapshot at real-time t is defined as the collection of the clock error estimations $\varepsilon_i(t) = t - LC_i(t)$, $\forall i \in V$ at time t. In our simulation, we take a snapshot at the end of each round as a performance measure for our clock synchronization algorithms. Note that only the root node determines the initiation of a new round. With a snapshot taken at time t we define the *average* error of the logical clocks as

$$\bar{\varepsilon}(t) = \frac{1}{n-1} \sum_{i=1}^{n-1} \varepsilon_i(t),$$

where we consider all the nodes in V except the root. We also define the *maximum* error as $\hat{\varepsilon}(t) = \max_{i \in V} \varepsilon_i(t)$.

Finally, in our experiments we observed that there is a round k after which the maximum error of logical clocks, $\hat{\varepsilon}$, stabilizes around a value $x \pm \xi$, where ξ is on the order of hundred nanoseconds. When this condition is achieved we say that the clocks have *stabilized*. Typically, in our simulations, this stabilization occurs after three rounds. In what follows, we base our comparisons and results on clock error profiles after this stabilization.

4.3 Opportunistic versus Tree-Based Clock Synchronization

Given Algorithms 1 and 2, introduced in Chapter 3, we compared them for the different topologies in Figure 4.1. Our simulation results show that opportunism in clock synchronization decreases the average clock error, $\bar{\varepsilon}$. Also, the size of the WOP reveals the introduction of an additional trade-off in the performance of Algorithm 2. In fact, this window should not be arbitrarily large. These conclusions, based on our experiments, are explained in the following paragraphs.

For Algorithm 2, we used different values of δ in our simulations. The minimum value was selected considering the network diameter of the topology and approximating the time needed to propagate a message through it. For example, for some stable rounds in topology 2, we observed that nodes received at most 1 message per round with $\delta = 0.5$ ms, despite having several neighbors; with $\delta = 2$ ms, some nodes were able to receive messages from all

their neighbors, during several rounds.

In order to compare the algorithms, we computed the average, $\bar{\varepsilon}$, and standard deviation of the errors of the nodes' logical clocks. These values were collected from several clock synchronization rounds as follows: **1)** once the system had stabilized, in terms of error values, we took snapshots until the end of the simulation; **2)** then, for each snapshot we computed $\bar{\varepsilon}$, and finally **3)** we took the average of all of these $\bar{\varepsilon}$. The reason we did not use a single snapshot after stabilization in our results is the behavior of Algorithm 1. In fact, the simulations revealed that despite the use of static topologies, for some rounds in Algorithm 1, some portion of the network remained unsynchronized due to packet collisions. Since nodes are supposed to correct their logical clocks only according to the messages received from their parents, packet losses are significant for clock synchronization in this case. Thus, a single round is not enough for gathering conclusions.

Figure 4.2 shows the average, $\bar{\varepsilon}$, and standard deviation of the logical clock's errors, in topology 1 (no traffic). Algorithm 2 exceeds the performance of Algorithm 1 (our baseline) for all values of δ tested in our simulations; in this particular case, $\bar{\varepsilon}$ decreases by about 17% with our opportunistic approach, while the standard deviation reduces by at least about 27%.

Next, Figures 4.3, 4.4, 4.5 and 4.6 show the performance of Algorithms 1 and 2 in topologies 2, 3, 4 and 5 respectively. For all cases, $\bar{\varepsilon}$ is decreased with Algorithm 2. However, we observed that the degree of reduction depends on the network density. Note that in topology 2, $\bar{\varepsilon}$ decreases by about 60%; then, in topology 3, $\bar{\varepsilon}$ is reduced by 40% and this value becomes smaller for topologies 4 and 5 as well. Recall from Figure 4.1 that, in general, the diameter of the network does not change when incrementing the number of nodes. Therefore, from topology 1 to topology 5, the number of nodes that belong to the neighborhood of a node v_i increases.

At high densities, wireless networks become much more likely to suffer communication failures due to contention for their shared communication medium; Ganesan *et al.* [26] reports a message loss of 20% and above between adjacent nodes in dense wireless networks. Therefore, for denser topologies, we believe that the higher levels of contention and collisions in the network affect the synchronization accuracy adversely. In fact, contention and collisions cause packet losses and extra delay in the transmission of messages, which may result in **1)** a node v_i failing to receive a synchronization mes-

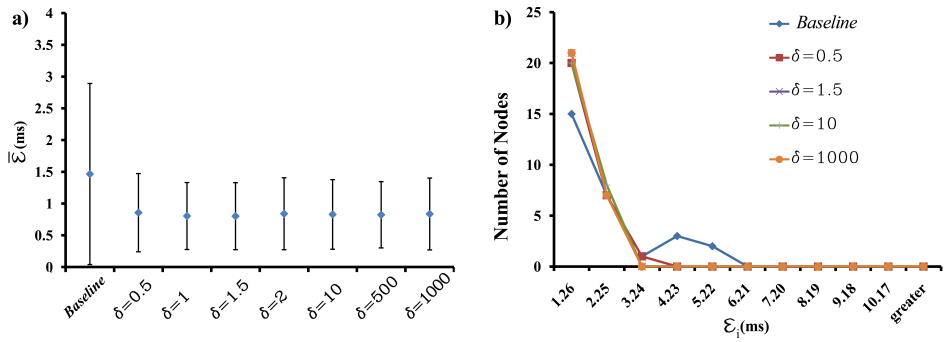


Figure 4.2: a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 1, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms.

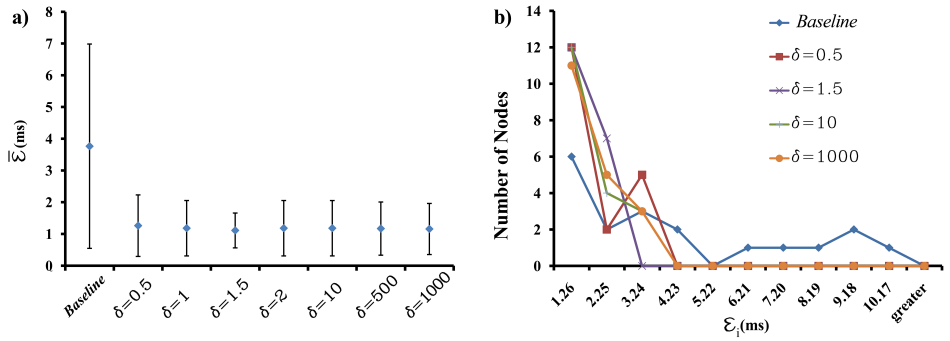


Figure 4.3: a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 2, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms.

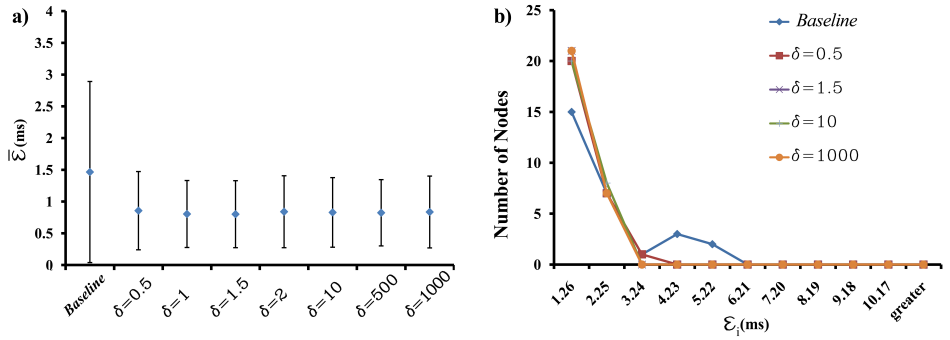


Figure 4.4: a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 3, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms.

sage during the WOP, and **2)** additional error in LC_i when it is corrected according to a delayed message with higher uncertainty.

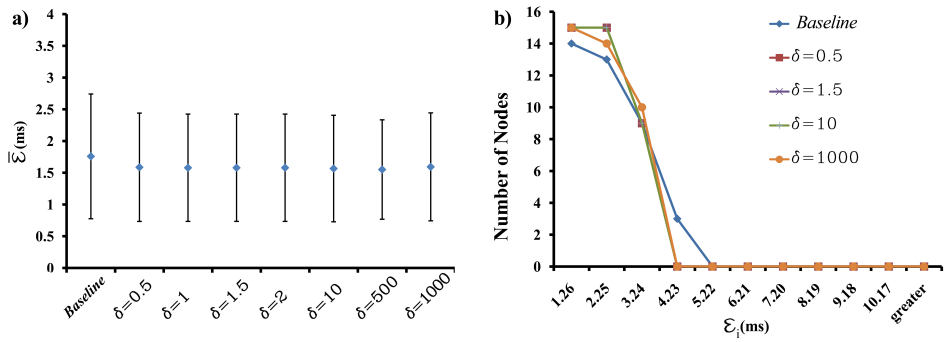


Figure 4.5: a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 4, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms.

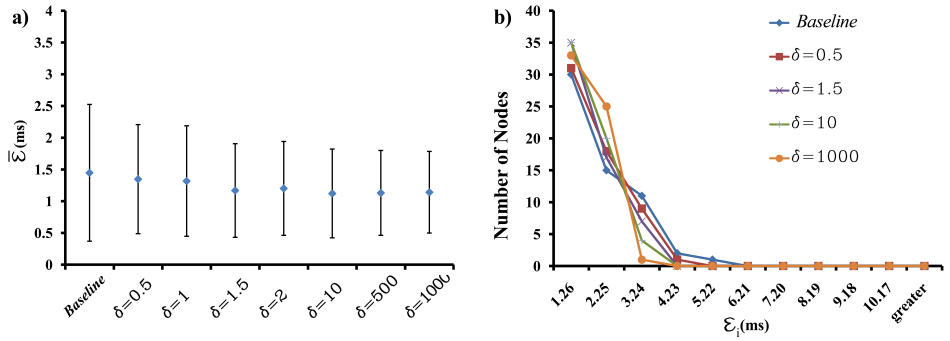


Figure 4.6: a) Average error values, $\bar{\varepsilon}$, for Algorithm 1 (baseline) and Algorithm 2 in topology 5, with different values of δ in milliseconds. b) Error distribution (histogram) for the different algorithms.

Additionally, we also observed a trade-off between the size of the WOP and the average clock error, $\bar{\varepsilon}$. First, as δ increases, $\bar{\varepsilon}$ decreases, meaning that nodes might have received more than one useful synchronization message and adjusted their clocks that many times.

However, δ cannot become arbitrarily large. Depending on the density of the network, node v_i will hear from a particular set of neighbors. Therefore, increasing δ above a certain value will not help v_i since the number of messages received will not increase any further. In addition, a larger δ might imply extra error coming from the nodes' skews. For example, assume that v_i corrected its clock for the last time at t_1 and the WOP expired at t_2; when v_i broadcasts its time-stamp, its logical clock has already ticked $t_2 - t_1$ seconds at its own rate σ_i, which might significantly amplify the error. This *extrapolation* of time by v_i can be detrimental. Therefore, variations

in the values of $\bar{\varepsilon}$ not only reveal how many messages a node could have used for correcting its clock, but also reveal errors incurred by the different time-stamps carried by these messages. Analysis of error distributions when using $\delta \gg 1000$ ms shows worse performance of Algorithm 2 compared to Algorithm 1. For example, when using $\delta = 5000$ ms in topology 2, the average clock error increased by 0.5 ms compared to the mean error of Algorithm 1.

4.4 Pure Opportunism versus Randomized Opportunism

First, we define *overhead* as the average number of messages that are broadcast in the network by the clock synchronization protocol per round. Algorithm 3 improves upon Algorithm 2 by decreasing the overhead in the network. For this experiment, we considered stable error profiles (i.e. one snapshot after stabilization), and we measured the number of messages that were *i)* sent, *ii)* received, and *iii)* dropped during the entire duration of the simulation.

We observed that for all the topologies in Figure 4.1, the overhead was decreased by Algorithm 3. The amount of reduction is also related to the density of the network. For example, in topology 1 there were only about 2% fewer messages when using Algorithm 3. Similarly, in topology 2, the reduction was around 4% of the total number of messages sent with Algorithm 2. For denser topologies, the reduction value increases: thus, the overhead was further reduced by about 24%, 27%, 36% in topologies 3, 4 and 5, respectively.

Randomized broadcasts might reduce the number of messages received per node during the WOP since some nodes remain silent during a synchronization round. This might increase the nodes' clock errors. However, we observed an interesting result: Algorithm 3 could even reduce the average clock error, $\bar{\varepsilon}$, in some cases. Figures 4.7, 4.8, 4.9, 4.10, and 4.11 show the values of $\bar{\varepsilon}$, resulting from Algorithms 2 and 3 in topologies 1, 2, 3, 4 and 5 respectively. In topologies 1 and 2, for every value of δ, Algorithm 3 performed better than Algorithm 2; while in the other topologies, Algorithm 3 performed better for some values of δ. For example, in Figure 4.9, Algorithm 3 decreased $\bar{\varepsilon}$ for three values of δ only.

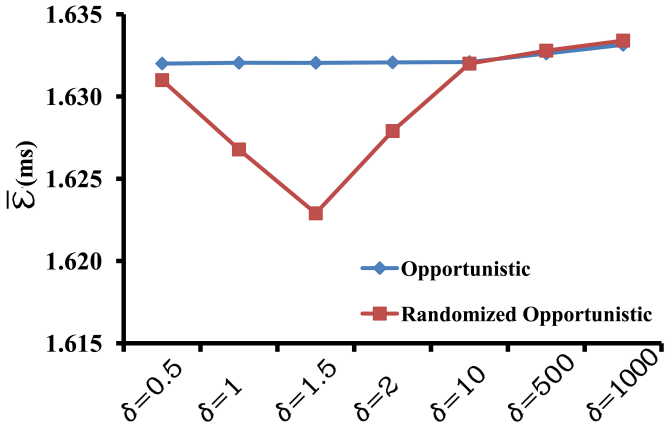


Figure 4.7: Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 1, for different values of δ in milliseconds.

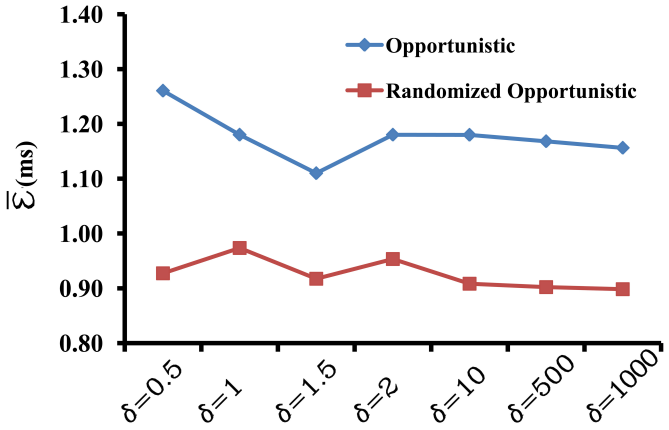


Figure 4.8: Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 2, for different values of δ in milliseconds.

In our simulations, we always observed that Algorithm 3 reduced the number of packet collisions, compared to Algorithm 2. Therefore, the behavior described in the previous paragraph can be explained as follows:

- As the number of packet collisions decreases, the number of synchronization messages received per node increases. For example, in topology 3, for $\delta = 1.5$ ms the number of dropped messages decreased by 35.56% for Algorithm 3 compared to Algorithm 2. In this case, some nodes were able to correct their clocks larger number of times during the WOP.

- A smaller number of collisions means reduced contention for transmit-

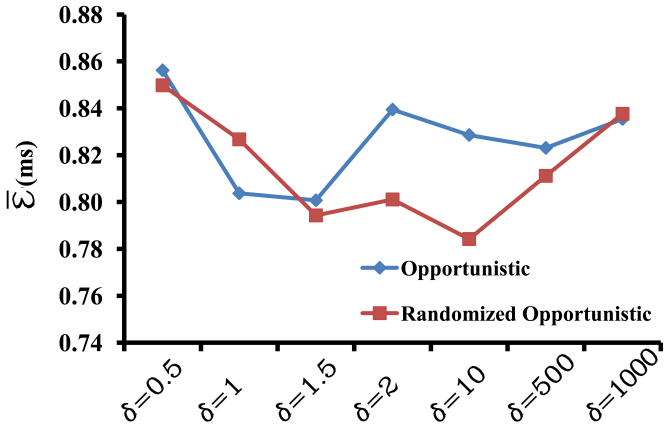


Figure 4.9: Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 3, for different values of δ in milliseconds.

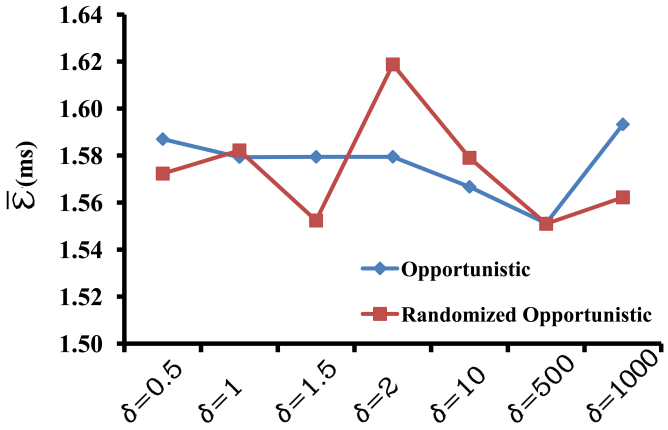


Figure 4.10: Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 4, for different values of δ in milliseconds.

ting on the channel. As a result, with Algorithm 3, nodes can transmit earlier than by using Algorithm 2. This has a direct impact on the accuracy of the synchronization, considering that we only rely on propagation delays. This was the situation exhibited in topology 1.

In summary, randomized opportunistic synchronization reduced overhead and even improved synchronization accuracy over the deterministic opportunistic algorithm and the baseline.

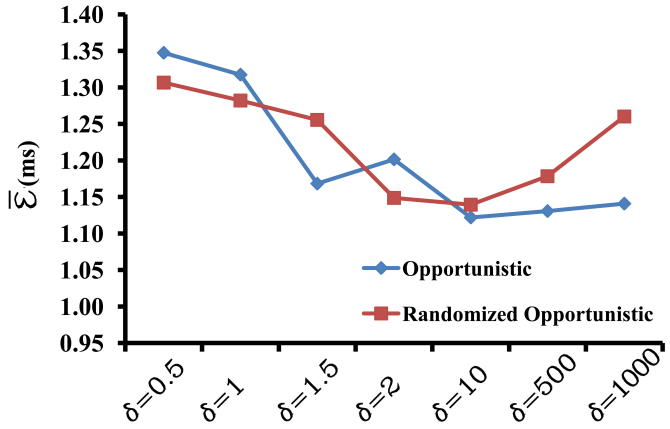


Figure 4.11: Average error values, $\bar{\varepsilon}$, for Algorithm 2 and Algorithm 3 in topology 5, for different values of δ in milliseconds.

4.5 Overhead Comparison

In order to evaluate the overhead incurred by Algorithm 2, we compare it against an existing clock synchronization protocol that also uses the concept of opportunism: the TSync protocol, proposed by Dai and Han in [13]. These two algorithms are similar in the following aspects:

1. They do not rely on a strict tree structure for propagating clock values.
2. A reference node initiates the synchronization process in every round.
3. They are both opportunistic in the sense that in every round, a node might correct its logical clock from more than one *sync* received message.
4. TSync uses the notion of *levels* in order to know how far away a particular node is from the reference node. Thus, as mentioned in Chapter 3, a node v_i corrects its clock when receiving a message from any node belonging to a lesser level than its own. Similarly, Algorithm 2 uses the number of hops as a decision metric for clock correction.

On the other hand, these two protocols are different because TSync uses the two-way message exchange in order to compute message delays and offsets. Instead, Algorithm 2 considers propagation delay only, for correcting logical clocks. Furthermore, in TSync, every time a node corrects its logical clock during round k, it will initiate a three-message transmission per

broadcasting domain, defined as the coverage area centered at v_i. In our algorithm, although v_i might also correct its clock many times per round, it will broadcast one message only, at the end of the WOP.

We illustrate the difference in the overhead of these two algorithms with an example. Figure 4.12 shows the topology on which we exemplify an execution of both: TSync and Algorithm 2. First, consider the following assumptions for our particular example:

- Node 1 is the root.
- There is no interference for message transmissions in the network (i.e. the message delay uncertainties computed using 1) two-way message exchange, and 2) propagation delay, are comparable).
- Clocks have the same skew for all nodes.
- WOP is chosen such that the number of messages that a node receives with Algorithm 2 is equal to the number of messages received by that node with TSync.

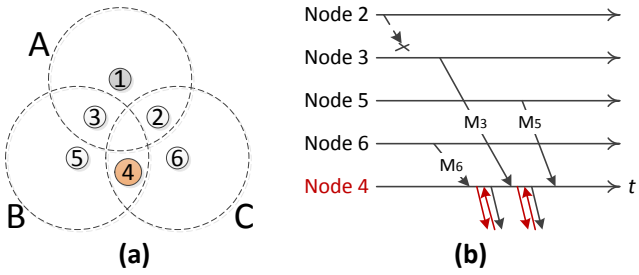


Figure 4.12: Illustration of overhead by using TSync on the given topology. (a) Small topology with three broadcasting domains A, B, and C. Node 1 is the root in the network and node 4 is the focus of our analysis. (b) Reception of messages at node 4 for a particular execution. Vertical lines are timelines illustrating the order of events, and oblique lines are messages transmitted from senders to receivers. Since TSync uses a three-message exchange for synchronization, a black line is the last message of this trio, which triggers clock corrections, and a red line is either the first or second message of it.

Now, let us show the behavior of both algorithms separately, by looking at the overhead generated at node 4 at a particular round, after node 1, the root, initiates the synchronization process in the broadcast domain A.

1. **TSync algorithm:** According to Figure 4.12, two messages triggered a clock correction at node 4: one from node 6 (2 hops away from the root) and one from node 3. Every time node 4 corrected its clock, it caused the exchange of 3 synchronization messages in its broadcasting domains B and C. Note that node 4 discarded the message from node 5, since it came from a two-hop path and node 4 had already received a message with that number of hops.

2. **Algorithm 2:** Assume that node 4 receives synchronization messages in the same order as in Figure 4.12. Then node 4 corrects its clock according to messages from nodes 6, 3 and 5. At the end of the WOP, node 4 broadcasts 1 *synchronization* message in its broadcasting domains B and C.

In this simple execution, with TSync, node 4 caused the transmission of 6 synchronization messages in its broadcasting domains in order to allow its neighbors to synchronize according to its LC_6. When using Algorithm 2, for the same conditions, node 4 only required 1 synchronization message in its broadcasting domains. With a larger topology, if we construct an execution where the situation just described repeats in other parts of the network, Algorithm 2 will decrease the overhead considerably, compared to TSync. Therefore, we consider our algorithm as a lightweight approach for clock synchronization.

We evaluated this behavior experimentally by comparing Algorithms 4 and 2 for topologies 2 and 4. We again considered the total number of messages sent during all the executions after stabilization, for both algorithms. As previously predicted, Algorithm 2 decreases the overall overhead. For example, in topology 2, around 47% of the messages sent with Algorithm 4 were decreased by using Algorithm 2; in topology 4, a denser network, the overhead was reduced by about 55%.

In spite of the improvement of our opportunistic algorithm in terms of overhead reduction, TSync provided smaller errors for the nodes' logical clocks. Figures 4.13 and 4.14 provide the average, $\bar{\varepsilon}$, and standard deviation values from a stable snapshot, which resulted when executing Algorithms 4 and 2 in topologies 2 and 4 respectively.

Note that, independently of the density of the network, TSync results in a small value of $\bar{\varepsilon}$. Thus, its extra overhead could be acceptable for applications

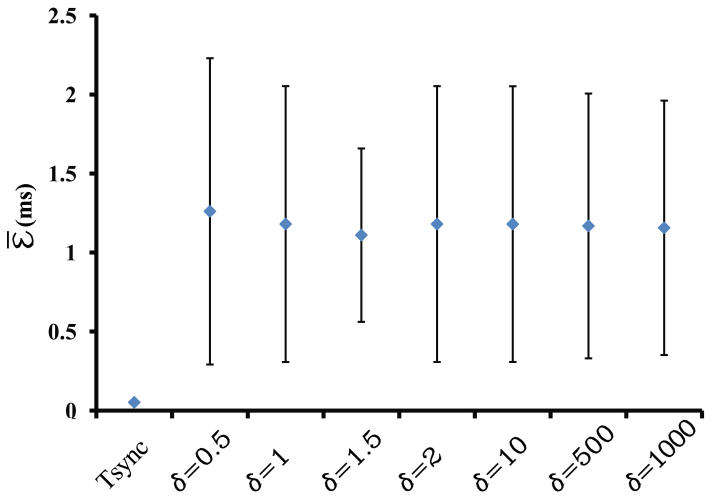


Figure 4.13: a) Average error values, $\bar{\varepsilon}$, for Algorithm 4 and Algorithm 2 in topology 2, with different values of δ in milliseconds.

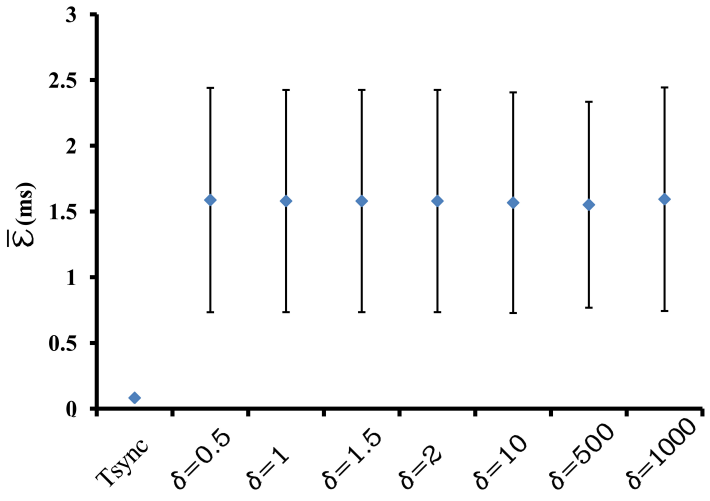


Figure 4.14: a) Average error values, $\bar{\varepsilon}$, for Algorithm 4 and Algorithm 2 in topology 4, with different values of δ in milliseconds.

that seek clock accuracy. However, less overhead means extra battery life, which is an important need for wireless ad hoc networks.

4.6 WOP and the Extrapolation Problem

Finally, we provide an intuition for overcoming the extrapolation problem mentioned in Section 4.3. For instance, node v_i does not necessarily need to wait until the end of the WOP to broadcast LC_i; in fact, v_i could wait until receiving m *sync* messages and then broadcast immediately.

For topology 2 and $\delta > 1.5$ ms, we evaluated the number of *sync* messages that v_i received during round k, and the accuracy of their associated time-stamps compared to the very first time-stamp received by v_i during the same round. For this analysis, we define *accuracy* as the difference between $LC_i(t)$ and real-time t. Therefore, a *sync* message received by v_i in round k is said to be a *better* message if the accuracy of LC_i, immediately after adopting the respective time-stamp, is less than the accuracy that LC_i had when it adopted the first time-stamp in that round.

We performed this analysis for the two traffic scenarios. For the entire duration of each simulation, we computed the total number of clock corrections performed, and the number of better messages received. For both traffic scenarios in this particular topology, around 50% of the total number of clock corrections were triggered by better messages. Moreover, by analyzing the distribution of these better messages, it turned out that around 60% of the third messages received had lower accuracy than the first received message, meaning that these messages might have come along a shorter route than the first one. From these results, for example, node v_i could decide to broadcast LC_i right after receiving this third message instead of waiting for the WOP to expire.

The implementation of this variation of Algorithm 2, and its experimental evaluation, remain for future work.

CHAPTER 5

CONCLUSIONS

For many applications of wireless ad hoc networks time synchronization is an inevitable component. Either for calibrating the data gathered from a sensor network, exchanging time-stamped messages, or disseminating synchronized commands in the network, clock synchronization is essential.

In this thesis we propose an opportunistic clock synchronization algorithm for ad hoc networks where, in each round, nodes may correct their clocks many times according to the duration of a window of opportunity (WOP). Thus, we rely on the basic idea of exploiting the broadcast nature of the wireless channel. Our protocol time-stamps messages at the MAC-layer, for reducing the uncertainty of message delays, and depends on a root node that provides a reference time that is propagated in a gossip-like fashion throughout the network.

When compared to a tree-based protocol (the baseline), where clock adjustments depend on specific nodes such as *parents*, the evident advantage of our opportunistic algorithm is that the probability of a node left unsynchronized during round k decreases considerably. The reason is that a packet collision from one source might be overcome with a successful reception from another source.

More importantly, experiments show that our opportunistic algorithm decreases the average error of nodes' logical clocks. However, the results also reveal that the degree of improvement of our protocol is related to the density of the network. For instance, we show that the accuracy of clock synchronization is negatively affected in denser topologies due to higher levels of contention and collisions. In IEEE 802.11 DCF, a back-off interval is implemented as a response to contention; however, this directly increases the uncertainty of message delays, which has been identified as the major source of error in clock synchronization algorithms.

To reduce collisions and contention in dense networks, we further decrease

the incurred overhead by implementing a variation of our opportunistic protocol, which uses randomized broadcasts. Thus, every node is assigned a probability for transmitting a synchronization packet, depending on the number of neighboring nodes. Surprisingly, our randomized opportunistic algorithm contributes to average error reduction since more packets are successfully received during the WOP due to fewer collisions in the network.

Finally, we show that our algorithm is lightweight compared to algorithms that also use opportunism, such as the TSync protocol. In fact, our experiments suggest that TSync is costly in terms of overhead; for example, in a specific topology it produced twice the number of messages sent by our opportunistic approach. However, TSync appeared to achieve better accuracy due to its method that accounts for message delay uncertainties. Time-stamping at the MAC layer did not reduce these uncertainties as accurately as the two-way message exchange, which may be attributed to the overhead introduced by the IEEE 802.11 MAC layer, as we mentioned above. In general, more message transmissions are expected (high overhead) in order to design more accurate clock synchronization algorithms.

Different applications have specific needs in terms of accuracy, power consumption, and complexity of the adapted clock synchronization scheme. A synchronization technique designed for wireless ad hoc networks should use few message exchanges, require small computational power and should have reasonable precision. These are the characteristics of our opportunistic clock synchronization protocols introduced through this research work.

CHAPTER 6

FUTURE WORK

In our proposed opportunistic clock synchronization algorithm, each node relies on a metric in order to accept or deny a synchronization message. The metric we use is the number of hops that the message has traversed from the root. However, this metric is not sufficient for determining whether a message is "good" or not. For instance, a route of 2 hops might be better than a single-hop route if the latter is exposed to high levels of interference. Therefore, for a particular topology it might be useful to study the reliability of each link (i.e. percentage of messages that were successfully received in a given period of time), so that a better decision metric can be established for our opportunistic algorithms.

In Chapter 4 we explained what we call *the extrapolation problem*, and we suggested an alternative algorithm that could potentially overcome it. The experiment we described should be further studied in different topologies in order to determine similarities among the distributions of *better* messages. Moreover, this variation of Algorithm 2 should be implemented so that its precision can be measured and compared.

Finally, it is necessary to analyze the duration of the WOP. We believe that an optimal duration might be determined based on the density of the network (i.e. the average number of neighbors that each node has) and the reliability of the links. As a result, with such an optimal WOP, the average error of our opportunistic algorithm could potentially decrease.

REFERENCES

[1] Department of Defense U.S. Air Force, "ARGUS Advanced Remote Ground Unattended Sensor Systems." [Online]. Available: http://www.globalsecurity.org/intell/systems/arguss.htm

[2] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281–323, May 2005.

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

[4] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 847–853, 1989.

[5] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Journal of the ACM*, vol. 34, p. 626–645, July 1987.

[6] J. Elson and K. Römer, "Wireless sensor networks: a new regime for time synchronization," *ACM SIGCOMM Computer Communication Review*, vol. 33, p. 149154, Jan. 2003.

[7] J. L. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, Apr. 1988.

[8] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, 2004.

[9] L. Subramanian and R. Katz, "An architecture for building self-configurable systems," in *2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, 2000, pp. 63–73.

[10] Internet Engineering Task Force, "Simple network time protocol (SNTP)," *IETF RFC 2030*, 1996.

[11] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, p. 138149.

[12] H. Attiya, D. Hay, and J. L. Welch, "Optimal Clock Synchronization Under Energy Constraints in Wireless Ad-Hoc Networks," in *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS)*, 2005, pp. 221–234.

[13] H. Dai and R. Han, "TSync: a lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, p. 125139, Jan. 2004.

[14] K. Fall et al., "The ns-2 manual," *The VINT Project*, April 2002.

[15] N. Vaidya, "Network-aware distributed algorithms: Challenges and opportunities in wireless networks," Presented at DISC 2010, Cambridge, MA, USA, September 2010. [Online]. Available: http://www.crhc.illinois.edu/wireless/talks/DISC2010_vaidya.ppt

[16] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, p. 147163, Dec. 2002.

[17] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "Robust multi-hop time synchronization in sensor networks," in *Proceedings of the International Conference on Wireless Networks (ICWN04)*, 2004, pp. 454–460.

[18] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, p. 3949.

[19] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* New York, NY, USA: Wiley-IEEE, 2004.

[20] S. H. Lee and L. Choi, "Chaining Clock Synchronization: an Energy-Efficient Clock Synchronization Scheme for Wireless Sensor Networks," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, 2009, p. 172177.

[21] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, 2000, pp. 125–132.

[22] L. L. Peterson and B. Davie, *Computer Networks: A Systems Approach*. San Francisco, CA, USA: Elsevier, 2007.

[23] I. Stojmenovi, *Handbook of Sensor Networks: Algorithms and Architectures*. New York, NY, USA: John Wiley and Sons, October 2005.

[24] N. Deo and C.-Y. Pang, "Shortest−path Algorithms: Taxonomy and Annotation," *Networks*, vol. 14, no. 2, pp. 275–323, 1984. [Online]. Available: http://dx.doi.org/10.1002/net.3230140208

[25] Y. Sasson, D. Cavin, and A. Schiper, "Probabilistic broadcast for flooding in wireless mobile ad hoc networks," in *2003 IEEE Wireless Communications and Networking Conference*, vol. 2, 2003, p. 11241130.

[26] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "An empirical study of epidemic algorithms in large scale multihop wireless networks," 2002. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.3915