Clock Synchronization for Multi-hop Ad hoc Networks

Neil O' Connor B.Sc.

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
M.Sc in Computer Science

2003

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Neil O' Connor B.Sc.

15^{th}September 2003

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Neil O' Connor B.Sc.

15^{th}September 2003

Acknowledgments

I would especially like to thank my supervisor Prof. Vinny Cahill for his guidance. His inciteful views and understanding made the process far more interesting and less painful than it might have been. Thanks to Adrian who worked with me in the early stages, and Ray who always had time for one more question.

Thanks to my family and friends, who understood when they didn't see me for weeks on end. And finally thanks to my classmates, whose friendship bore me over the lows when there seemed to be no end in sight.

Abstract

Clock synchronization is one of the basic requirements of a distributed real-time system. The greater the precision achievable by the clock synchronization algorithm used, the harder the real-time guarantees that can be given by the system. Numerous clock synchronization algorithms for wired networks have been proposed, which provide varying degrees of precision.

Wireless networks provide a number of challenges which clock synchronization algorithms for wired networks do not address satisfactorily. Dynamic network tolopogy and the existence of multi-hop neighbours, as well as high message loss make wired clock synchronization algorithms unsuited to the wireless environment.

This work presents a clock synchronization algorithm for multi-hop ad hoc networks, which improves on the precision achievable by other such algorithms while minimising the prerequisites of the algorithm. Unlike other algorithms which provide clock synchronization in wireless networks, this algorithm does not require a centralised master node, or predefined slots in which nodes can broadcast their synchronization messages. It utilises the *tightness* property of the wireless medium, whereby message receipt occurs at the same physical instant across all recipients. The problem of multi-hop synchronization is addressed by using local averaging at overlapping subsets of the nodes to achieve global synchronization.

A theoretical bound on the achievable precision for one-hop synchronization has been calculated. This bound applies in the absence of lost messages and network partitions, with the precision returning below the bound upon merging the partitions. The upper bound on precision for one-hop synchronization extends to n times the bound for n-*hop* synchronization in a multi-hop network. The operation of the algorithm has been simulated to verify this bound.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Clock synchronization is one of the fundamental problems of distributed computing. There are numerous applications for synchronized clocks in distributed systems [10]. These vary from version and concurrency control using timestamps in distributed database systems, to resource allocation and communication protocols which rely on timeouts. Communication protocols which use time division to provide quality of service (QOS) guarantees for access to the transmission medium, also rely on a global time base. These protocols are required for video-on-demand and group conferencing systems. The performance of these schemes depends to a large degree on the quality of synchronization provided.

A distributed system is often expected to carry out some synchronized task - to co-operate in real-time. Decentralized agreement as to when events occur or when to trigger specific actions can be achieved if each processor has the same local time. The use of distributed algorithms which proceed in rounds, and are hence simpler to design, can then be enabled[10].

The general problem of clock synchronization occurs because of the unreliable nature of the local clocks used on computers in a distributed system. These local clocks tend to drift apart over time. Therefore the goal of clock synchronization is to ensure that physically dispersed processors share a common notion of time, using local clocks to maintain the time and exchanging messages to coordinate their views of the time. The local clocks may be subject to varying drift rates, and the communication medium over which messages are passed is subject to unknown transmission times [7].

1.2 Clock Synchronization for Wired Networks

Clock synchronization algorithms for wired networks have developed to provide increasing quality of synchronization. The limitations which affect how closely clocks can be synchronized have been reduced from the round-trip delay [2, 13], to the propagation delay [4, 13, 15] and to the variance in propagation delay [16]. The advent of the Global Positioning System (GPS) has greatly changed the situation. This provides an accurate and inexpensive means of acquiring accurate timing information using a radio receiver. A device consisting of nothing more than a GPS receiver and a network interface can be placed directly on a shared communication bus, providing the nodes on that bus with periodic time packets [7]. This may well signal the end for traditional clock synchronization algorithms for wired networks.

1.3 Clock Synchronization for Wireless Networks

Wireless networks introduce additional demands on clock synchronization algorithms. Increased interest in the area of ad hoc networks has given rise to greater demands for clock synchronization. Ad hoc networks are sets of mobile nodes which communicate wirelessly. Nodes which are not directly in range can communicate via chains of nodes which lead from the source node to the destination node. Each of the links in a chain is hereafter referred to as a *hop*.

In addition to traditional distributed system requirments for clock synchronization, new application areas such as mobile robotics and gaming place the emphasis on the timely delivery of data. The availability of a global time base is key to providing time bounded access to the transmission medium and delivering quality of service guarantees.

Assumptions made by traditional wired synchronization algorithms, such as direct connection to a timeserver, cannot be guaranteed for wireless environments where multi-hop routes link nodes of the network together. GPS does not provide all the answers to the problem either. Relying on a wireless node always in range of a GPS time server would be over-restrictive on the movement of the nodes. Equally, the assumption that each node has access to a GPS receiver of its own doesn't solve the problem, as the mobility of the nodes means that they may at some stage be out of direct line of sight of GPS satellites. This means that situations will arise where GPS enabled receivers will not be able to provide clock synchronization.

A number of algorithms have been proposed specifically for clock synchronization in wireless

networks. The algorithms described in [6, 11] provide clock synchronization for infrastructure mode 802.11 [6] radio communication. The solution in [11] manages to remove the propagation delay completely from the time-critical path. These algorithms use a centralised access point which is synchronized to by all ad hoc nodes. This solution is limited by the transmission range of the access point, and so is not suitable for a multi-hop ad hoc environment.

Clock synchronization for multi-hop wireless networks is provided by the algorithms in [14, 1]. These algorithms both use the idea that local averaging at overlapping subsets of the nodes will lead to global synchronization. While multi-hop synchronization is achieved, the achievable synchronization quality is not of the same standard as some other algorithms and this is addressed by the presented algorithm.

1.4 An algorithm for Multi-hop Ad Hoc Network Clock Synchronization

The proposed algorithm aims to provide a higher quality of clock sychronization than other algorithms for multi-hop ad hoc networks. This is achieved by exploiting the *tightness* property of the medium as in [11]. Tightness is the assumption that the same message is received simultaneously across all receivers. A novel way of exploiting this property for wireless networks is used. Receivers of the same messages compare their timestamps for a particular message receipt. The synchronization achievable is limited only by the difference in processing time when receiving and timestamping a message.

The algorithm uses the same method of achieving multi-hop clock synchronization as in [14, 1]. Local averaging of overlapping subsets results in global synchronization. This achieves the dissertation goal; an algorithm which improves the synchronization precision achievable for multi-hop ad hoc networks.

1.5 Document Roadmap

Chapter 2 provides a state of the art review of existing clock synchronization protocols - how they address the goals stated at the start of the chapter, and their applicability to the problem at hand.

Chapter 3 looks at a number of solutions which were applicable to the problem of clock synchro-

nization for multi-hop ad hoc networks.

Chapter 4 covers the design and implementation of the simulator which modelled the operation of the protocol.

Chapter 5 analyses experiments carried out on the simulator. An evaluation of the algorithms performance with and without packet loss is presented.

Chapter 6 contains the conclusion to the work and future directions in which it might be extended.

Chapter 2

State of the Art

There are a number of goals which are addressed by clock synchonization algorithms.

- **Precision**

  The main goal of a clock synchronization algorithm is to achieve agreement between a set of clocks. This is referred to as the *precision* of the algorithm. *Precision is a characterisation of the degree to which any pair of correct clocks can differ over time [7].*

- **Accuracy**

  Accuracy is a characterisation of the degree to which a correct clock can differ from an external clock that gives the true time [7]. This may be expressed as an absolute bound between a clock and Physical time, or as a maximum drift rate of a clock from Physical time.

- **Minimizing Messages**

  Minimizing the number of messages exchanged is important where available bandwidth is a concern, for example in the area of wireless communication. This is therefore another valid concern when designing a clock synchronization algorithm.

- **Clock Adjustment**

  The method by which an adjustment is applied to the local clock may be an issue. Processes may require that the clock is never set back, or that large adjustments are not made.

- **Faulty clocks**

  Values received from faulty clocks may have to be detected and discarded.

- **Byzantine clocks**

  The presence of Byzantine (two faced) clocks may have to be dealt with by the algorithm. These clocks can maliciously give different views of their clock value to different nodes. Therefore some means of determining consistency may be required.

It is important to draw the distinction between Physical time and the local time which exists on a node. Physical time is continuous and cannot be adjusted, and so it is the same for all nodes in the network. This idea becomes important as discussion turns to the Physical time at which nodes observe events in the network.

Local time is individual to each node and is maintained on the local clock of that node. It is affected by the drift rate of the local clock, and also by resynchronizations which occur on that node. The goal of a clock synchronization protocol is to achieve agreement between the local clocks of the network as to what their local time is.

Offset is the term used to describe the difference between two local clocks at some point in time.

2.1 Steps of a Clock Synchronization algorithm

An algorithm for clock synchronization can be divided into three steps:

- **Reading a remote clock**

  In order to evaluate where a node's clock lies with regard to the global time of the system, it is necessary to acquire some knowledge of the values of remote clocks in the network. This is referred to as reading a remote clock.

- **Calculating the local clock adjustment**

  Once remote clocks have been read, the adjustment to make on the local clock must be determined. While some algorithms are obvious in how they calculate clock adjustments, others can be quite complex e.g. detecting and discarding faulty clock readings.

- **Applying the local clock adjustment**

  The adjustment calculated must be applied to the local clock. This adjustment can be applied either instantaneously or continuously across an interval and is discussed in Section 2.6.

The precision which a clock synchronization algorithm can achieve is limited by two factors: clock reading error and physical clock drift. As already stated, precision is the measure of how closely

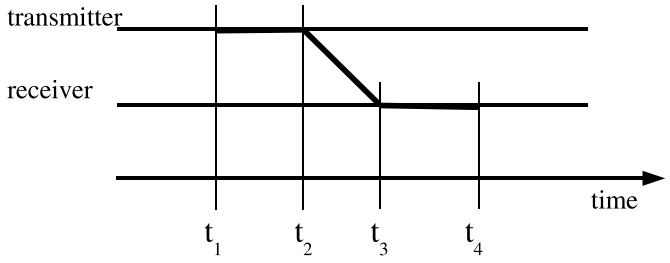


Figure 2.1: Propagation delay in the time-critical path

the synchronized clocks agree.

2.2 Clock Reading Error

The more precisely a remote clock can be read, the better the achievable synchronization precision becomes. There is a degree of uncertainty as to how 'old' a clock reading value if when it is used. Therefore minimising the clock-reading error is an important goal of any algorithm. The time-critical path is of particular interest when examining clock synchronization algorithms as it is a measure of the clock reading error. The time-critical path is composed of propagation delay for messages and also processing time, which limit the quality of the clock synchronization error by introducing uncertainty as to how long they take.

2.2.1 Propagation delay

A transmitted message is subject to some propagation delay, the size of which is unknown but greater than zero. This is the time which it takes for a message to travel from its origin to its destination. If we assume there is a maximum transmission time, then a timestamp created by a transmitter which is forwarded to another node, could potentially have aged by the maximum transmission time upon receipt. There is a degree of uncertainty here which results in its inclusion in the time-critical path (Fig. 2.1).

2.2.2 Processing time

Processing time can exist in two ways in the time-critical path. Processing time is the Physical time between an event occuring and that event being processed e.g. a point in time being reached,

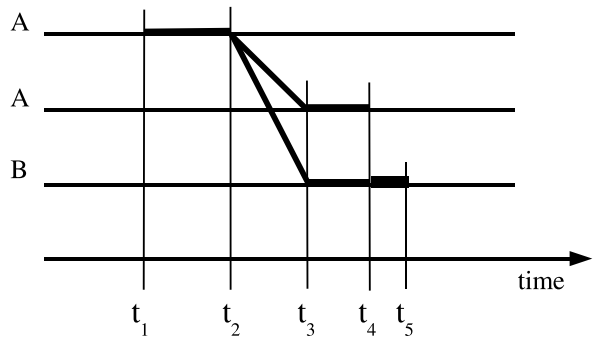


Figure 2.2: Processing time in the time-critical path

Algorithm 1 Worst Case Difference in Processing Time

$$max(proc) - min(proc)$$

where *proc* is the process time

or a message being received. When a process is run, the length of processing time is to some degree unknown. Speed of processor and scheduling can both affect the Physical time taken. Generally the processing time is simply added to the time-critical path ($t_2 - t_1$ in Fig. 2.2).

A special case is where processing is carried out in parallel on separate processors. This is of concern, for example, when taking timestamps for the an event. The difference in the physical instant at which the value of the local clock is taken on each processor, is important when calculating clock offset based on those timestamps.

It is currently impossible to provide predictable processing time on different processors. Real-time operating systems provide guaranteed response times to within a certain degree. This means that rather than the processing time as a whole being of concern, only the difference between the maximum and minimum bounds is inserted in the time-critical path (Alg. 1, ($t_5 - t_4$) in Fig. 2.2), as opposed to the entire processing time as is the general case.

2.3 Physical Clock Drift

Physical clock drift is the reason why clock synchronization algorithms are required. Physical clocks, even from the same manufacturer, do not maintain pace with each other. The only guarantee with a clock is that it will not stray by greater than some specified drift. During the period between

Algorithm 2 Worst Case Physical Clock Drift

$$2\theta\sigma$$

θ is the period of time
σ is the maximum drift rate of the Physical clock

resynchronizations, it is possible that the physical clocks will drift apart resulting in less precise synchronization. In the worst case this will result in 2 clocks drifting apart at the maximium drift rate across a period. (Alg. 2).

2.4 Wired Network Clock Synchronization

Many clock synchronization approaches have been proposed which address clock synchronization for wired networks. Since Lamport's landmark paper [8], numerous attempts have been made at achieving tighter synchronization between distributed clocks in a system. These algorithms can be segregated based on the propagation delay which affects their time-critical paths

2.4.1 Algorithms limited by Round-trip Delay

One of the most commonly used clock synchronization algorithms is the Network Time Protocol (NTP) [13]. NTP was designed to address the problem of clock synchronization across the Internet. Precision is achieved by ensuring nodes are accurate to Physical time, which leads to them being synchronized to eachother. It attempts to provide accurate synchronization in spite of variable operating speeds and reliability of network. The primary concern of NTP is to maintain accuracy between local clock time and physical time. This is accomplished by defining time servers and providing them with reliable time sources. Reliability is addressed by providing redundant time servers, and because NTP is designed to be deployed over large networks, diverse transmission paths are available. These serve to make it more likely that an operating time server can be reached. Messages can be authenticated using a checksum encrypted with the Data Encryption Standard (DES). Heterogeneity has to be addressed due to the multitude of different systems operating on the Internet, and to this end connectionless commmunication over the User Datagram Protocol (UDP) is used. Besides being a widely used standard, using connectionless communication also serves to minimize latency.

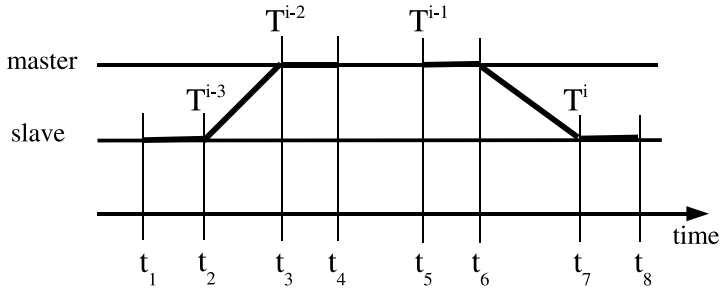


Figure 2.3: Network Time Protocol (NTP)

NTP uses a client-server architecture where a client uses timestamps from the server to synchronize to it. Clients and servers exchange periodic update messages these timestamps. The protocol can operate in one of three modes.

- **Multicast mode**

  Periodic broadcasts are made by the server to the local network, the nodes of which then synchronize to the server.

- **Precedure call mode**

  Clients request timestamps from the server

- **Symmetric mode**

  Nodes can be synchronized to by clients, and themselves be synchronized to another server.

The timestamps shown in Figure 2.3 are used as follows:

- $a = T_{i-2} - T_{i-3}$
- $b = T_{i-1} - T_i$
- Round trip delay $\sigma = a - b$
- Offset $\theta = (a + b)/2$

Each NTP message contains the latest three timestamps exchanged between a client and server, with the final timestamp being created on receipt of that message. The result is that each message is all the information required for a client to synchronize to a server. This addresses the issue of reliability, and message order.

Rather than all clients synchronizing to one server, NTP uses a self organising hierarchy to structure servers into a tree. Primary servers are defined which have access to a reliable time source. Once a client has synchronized to a primary server, it becomes a server to other nodes and identifies itself as being at one level lower in the tree than the server it synchronized to. In this way, where a number of servers are available for a client to synchronize to, a server is chosen which is the least number of levels below the primary server. The reason for this is that the closer a server is to the root of the tree the better a source of synchronization it is, as it is more precisely synchronized with the primary server.

When a node decides to resynchronize, NTP makes use of the fact that there are possibly a number of time servers available for a client to synchronize to. Having calculated offset values for each of those servers available, it uses a consistency algorithm to classify clocks as trusted or not based on values such as mean and variance. When a set of statistically equivalent clocks is reached, either the offset of one of these is chosen or their offsets are combined using a weighted-average algorithm based on their estimated errors. An assumption is made here that valid clocks are numerous relative to the number of faulty clocks in the nework, in order that faulty clocks can be statistically discarded.

If the time-critical path of this algorithm is examined (Fig. 2.3), it can be observed that the the Physical time between t_1 and t_4, and also between t_5 and t_8 is unknown. This results in the round trip delay being a part of the time-critical path of the algorithm. In addition the processing time surrounding each message transmission is included in the critical path. When a resynchronization occurs, the synchronizing process chooses the message with the smallest round trip delay from the last period and uses the information gathered to calculate the server clock's difference from the local clock. Choosing the fastest exchanged pair of messages from the last round minimizes the time-critical path.

NTP makes a number of assumptions which affect its applicability to the problem domain. The assumption is made that primary servers have access to a reliable external time source, and also that a client has access to time server. It is immediately obvious that because of the dynamic nature of wireless networks, it would be folly to assume that a node would always be in range of a server. For the same reason, a centralised solution would be very difficult to make fault tolerant as redundant servers might not be in range when the primary server fails.

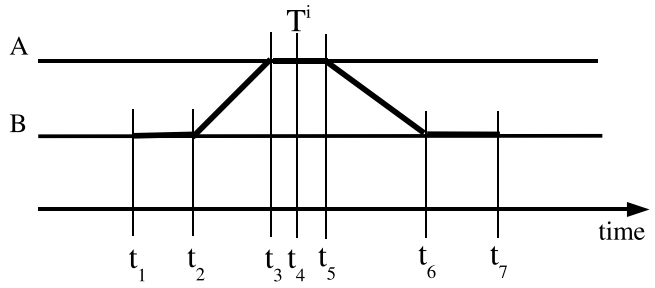


Figure 2.4: Probabilistic clock reading

An algorithm is proposed by Cristian in [2] addresses the problem that each attempted reading of a clock may not be successful. In addition, it succeeds in placing an upper bound on the clock reading error despite unbounded message delay. Clocks can be read directly or via other clocks which have an notion of the target clock's time. In order to reduce network overloading, synchronization messages are staggered across the period between resynchronizations.

As can be seen in Figure 2.4, a node makes a request for a timestamp from a peer. The time $t_7 - t_1$ taken to receive a reply is then observed. Based on this value, the clock reading operation fails or passes. If the time taken exceeds a predefined maximum then another request is sent in an effort to read that clock. After a predetermined period of retrys a clock reading failure occurs if no valid reading has been made.

A node can be in one of three states:

- **Request state**

  A node stays in this state as long as it has not read all other remote clocks in the network.

- **Reply state**

  A node remains in this state as long as it's clock has not been read by all other nodes in the network.

- **Finish state**

  A node moves to this state when is has read and been read by all other nodes in the system. Once a node has entered this state, it becomes a source of information about the time on all nodes in the network i.e. any node can use the finished node to acquire an approximation for any other clock in the system. This is known as *transitive* reading of remote clocks.

The algorithm addresses the problem of being unable to read a remote clock. This situation would result in different nodes reading different subsets of the global population of clocks, and thus adjusting their clocks inconsistently. A theoretical bound is placed on the difference between the values which a clock can be adjusted by, despite the fact that the adjustment may be based on different clock readings. A *midpoint convergence function* achieves this by making the assumption that all clocks are within a fixed value δ of eachother at initialisation, and that there is a maximum drift rate σ for physical clocks. This enables a formula which places a bound around the midpoint of any possible subset of the clocks, and also how far that midpoint could drift between resynchronizations. The assumption is made that resynchronization brings the values of the clocks back within the bound δ.

As with NTP, the time-critical path the round trip delay (Fig. 2.4). The length of the transmission time in each direction ($t_3 - t_1$ and $t_6 - t_4$) is unknown. The fastest message between processes is again chosen to minimise this bound. Due to the predefined maximum acceptable round trip delay, an upper bound exists on the length of the time critical path.

The configuration which this algorithm requires is an impedence to its use in wireless network clock synchronization. Both the acceptable time-critical path length and the time to spend requesting timestamps must be defined. The changing topology of an ad hoc network makes the time-critical path lenth unpredictable. Values which are too low will result in clocks not being read, while values which are too high reduce the effectiveness of bounding the time-critical path. The notion of transient clock reading could be useful for reading the clocks of ad hoc nodes which are out of direct transmission range, but which can have their clocks read via neighbouring nodes.. The midpoint convergence function relies on the assumption that on resynchronization, the maximum difference between clocks is always less than δ. For ad hoc networks, where network partitions fequently occur, this would not always be the case, as multiple resynchronizations might occur before the partitions were merged, resulting in the precision being above δ. In addition, the high message loss which is inevitable in a wireless environment would make it costly to repeatedly attempt to read a remote clock.

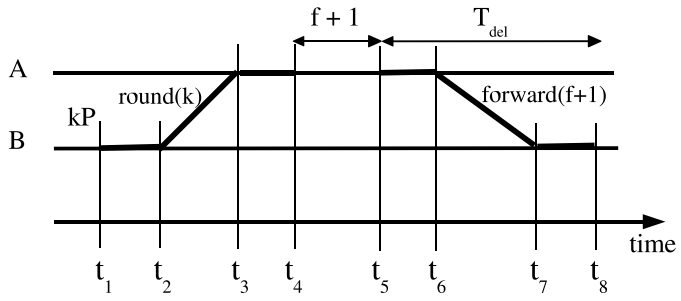


Figure 2.5: Optimal Clock Synchronization

2.4.2 Algorithms limited by Propagation Delay

The algorithms presented in [13, 2] and described in Section 2.4.1 are limited by the unknown round-trip delay. By removing the dependency of the algorithm on round-trip delay, and replacing it with propagation delay, the clock reading error is reduced and thus the synchronization precision which the algorithm can achieve is improved. In addition, this genre of clock synchronization algorithms are more fault tolerant due to their non-blocking qualities i.e. messages do not require a response

One of the first algorithms of this genre was presented by Srikanth et al in [15]. This algorithm aims to achieve precision between clocks of the network and also accuracy of the synchronized clocks with respect to real time. The goal of making the synchronization optimal is to achieve a drift rate for the synchronized clocks which is as little as that specified for the hardware clocks underlying the system. Up to f faulty processes are handled by the algorithm.

A fixed period P exists between resynchronizations (Fig. 2.5). When a node reaches the start of the k^{th} round kP (according to the value on its local clock), it broadcasts a *round(k)* message to inform other nodes of this. Once $f + 1$ of these messages have been received by a node, where f is the maximum number of faulty processes, the node knows that at least one correct process is ready to resynchronize. The node then starts its clock for the next round to be $kP + \alpha$, where α is a value chosen as being greater than the time taken to receive the $f + 1$ messages. α is added by each node when starting its clock to ensure that a clock is not set backwards. Once the clock is started for the next round, the $f + 1$ messages are forwarded so other nodes can start their clocks.

The algorithm makes the assumption that there is a fixed upper bound T_{del} on the time required for a message to be prepared, transmitted, received and processed ($(t_8 - t_5)$ Fig. 2.5). Also the

assumption is made that all messages are delivered successfully. These assumptions allow the fundamental idea upon which this algorithm is based. Due to all other nodes processing the $f+1$ messages within T_{del}, all other processes start their clock after at most T_{del}. Therefore the upper bound on the synchronization precision is T_{del}.

Providing optimal accuracy is also based on these assumptions. There is an uncertainty of T_{del} in the time it takes a process to accept a message. This uncertainty introduces a difference in the logical time between resynchronizations. The fastest clock resynchronizes every $P-\alpha$, while the slowest resynchronizes every $P-\alpha+T_{del}/(1+\rho)$, where ρ is the maximum drift of a physical clock. This is compensated for by delaying the start of the clock for the k^{th} round by $T_{del}/2(1+\rho)$ if a process receives a *round(k)* message early (according to its local clock), or advancing the start by the same amount if it receives a *round(k)* message late. The logical time between resynchronizations becomes $P-\alpha+T_{del}/2(1+\rho)$. This allows the drift of the population from Physical time to be bounded by $(1+\rho)$ above and $(1+\rho)^{-1}$ below.

The time-critical path of this algorithm ((t_8-t_5) Fig. 2.5) shows that because the *f + 1* messages are being forwarded but not replied to, they are subject only to the propagation and not the round-trip delay. This improves the synchronization precision of the algorithm over those described in Section 2.4.1. The dynamic topology of an ad hoc network makes it impossible to guarantee the timely delivery of a message to all nodes within T_{del}. Without the ability to make this assumption true, the algorithm becomes unworkable as a solution to the problem at hand.

2.4.2.1 Event Observation Algorithms

Event-observation algorithms relies on the observation and timestamping of events which occur. These events are usually message transmissions and receipts. The following algorithms make use of broadcast networks where it is assumed that message receipt is *tight* [16]. This basically means that while there is a propagation delay between the transmission and receipt of a message, the physical receipt time of the message is the same across all recipients.

The information disseminated among nodes in the network is an event. The local time of the event is noted, as opposed to the algorithms of Section 2.4.1 which return the local time upon a *read-clock* request, and are thus subject to the propagation delay on the return journey. The message has to travel in only one direction before timestamping occurs.

Rodrigues and Verissimo were the first to exploit the tightness property of a medium in [16].

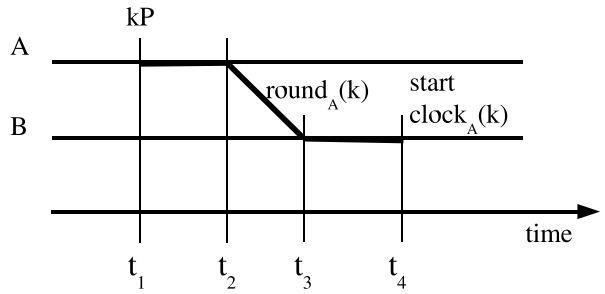


Figure 2.6: A Posteriori Synchronization

The algorithm handles incomplete message reception of broadcasts by all nodes in the network, and a distributed agreement protocol provides fault tolerance by eliminating a centralised decision making process.

A fixed period exists between resynchronizations. When a process reaches the resynchronization point according to its local clock value, it broadcasts a *Start(node_ID,round)* message, where *node_ID* uniquely identifies the node and *round* the round which the node has reached. The node then starts a tentative clock for the next round which it associates with its own node_ID. Upon receipt of a *Start(node_ID,round)* message from another process, each receiving process also starts a tentative clock with the value for the start of the round, and associates the tentative clock with the node_ID of the transmitter. The receipt of the message is then acknowledged in broadcast form, including the value of the nodes current clock for the last round (not the new tentative clock). The acknowledgement informs other nodes in the network which nodes have received which *Start(node_ID,round)* messages. A *Start(node_ID,round)* message which is acknowledged by all nodes in the network is termed a *simultaneous broadcast*. A distributed agreement algorithm is used at some later stage to select which of the tentative clocks, that were the result of a simultaneous broadcast should be the new clock for the next round.

In order that the algorithm stay approximately close to Physical time, the following procedure is carried out. After the tentative clock which will become the new clock has been chosen, the originator of the *Start(node_ID,round)* message chooses the median clock from the replies which it received, and uses this value to calculate the offset of this clock from its own clock. This is relying on the assumption that the median clock of the set will reflect most accurately the Physical time. The offset of the originator from this median value is then used to adjust all clocks to conform to what is assumed to be the best Physical time.

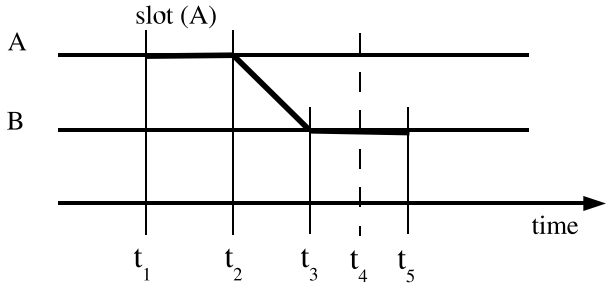


Figure 2.7: Time Distance Capture

Upon analysis of the time-critical path of the algorithm (Fig. 2.6), it can be observed that the earliest start of a particular tentative clock is on the node which transmits the *start(node_ID)* message for that clock (t_1). All recipients will start their tentative clocks after the message has been received, which is $t_3 - t_2$ time units later. This means that the clock start times are bounded by the propagation delay plus the processing time variance.

The assumption which requires that a simultaneous broadcast is achieved eliminates this algorithm as a possibility for multi-hop synchonization. Multi-hop neighbours would not observe the same events that start the tentative clocks and therefore resynchronization would never occur. In addition it must known which nodes should hear (and acknowledge) a broadcast in order to term it a synchronous broadcast. This knowledge would be difficult to provide in an ad hoc network where membership is constantly changing, and network partitions are the norm and not the exception.

Another algorithm which takes advantage of the tightness of the medium is described in [4]. The proposed algorithm uses predefined message transmission times in order to read remote clocks. The solution is designed for a multi-cluster system, where a cluster is a set of nodes. These share a common communication channel (where message receipt is tight). Clusters are joined together by gateways. The algorithm provides both precision and accuracy of synchronization.

The basic operation of the algorithm is as follows. Nodes in a cluster have a table of predefined transmission slots for each node in the cluster. A node transmits a message in it's slot according to its local clock (at t_1 Physical time) (Fig. 2.7). A receiving node expects to observe the transmission of A at time t_4, but it doesn't actually receive and process the message until t_5. By calculating the difference between t_4 and t_5, a receiver can calculate its offset from a transmitter. This is known as *one-shot* synchronization, as the receipt of a single message from the transmitter provides a node

with the transmitters clock value.

A subset of the nodes in a cluster are predefined reference nodes. The offsets of a node's local clock from these reference clocks are stored until its next resynchronization point. When resynchronization occurs, a fault tolerant averaging algorithm is used to discard the readings of faulty clocks. The values at each extreme of the range of values observed are discarded as faulty. The remaining values are then averaged and used to adjust the local clock. As all clocks observe and discard the same clock values, they set their clocks to the same average.

Clusters are joined together by a *gateway*. While the method described above uses a distributed algorithm, for multi-cluster synchronization the gateway is an authoritarian process which other nodes synchronize to. The clock of the gateway is read in the same way as above, with a predefined message transmission time for the gateway, which is observed by the nodes of the cluster. On resynchronization, the nodes adjust their clocks to synchronize to this value. Accuracy is maintained by linking the value of a gateway to an reliable external time source such as the Global Positioning System (GPS). This changes the status of the gateway to that of a *time gateway*.

The time-critical path of this algorithm can be seen in Figure 2.7. The Physical time between the transmitting node's intended start to its round (t_1) and the receiving node's observation of the transmission (t_5) is the time-critical path.

There are a number of similarities between the operation of clusters and those of ad hoc networks, which would suggest that this algorithm might be useful. A cluster is akin to a node's set of single hop neighbours. If this set of nodes were assigned transmission slots, then their offsets could be calculated in the same manner as used by this algorithm. However this solution assumes total connectivity via gateways, which link the clusters together. A solution for multi-hop ad hoc networks must assume partitions will occur. In addition, the gateways are predefined in this solution. In the dynamic layout of an ad hoc network, the gateway between clusters may change. Identification of the node which should act as an authoritarian process would prove a difficult hurdle to overcome.

2.5 Wireless Network Clock Synchronization

Several solutions have been proposed for clock synchronization in wireless networks.

The 802.11 standard for wireless communication ([6]) describes a clock synchronization algorithm for its infrastructure mode. In this mode all communication between nodes takes place through a centralised access point. The access point acts as a master to which the wireless nodes synchronize.

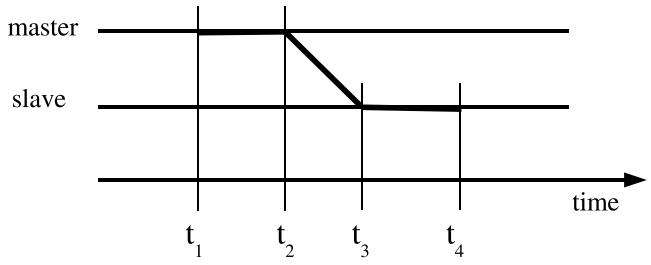


Figure 2.8: 802.11 Clock Synchronization

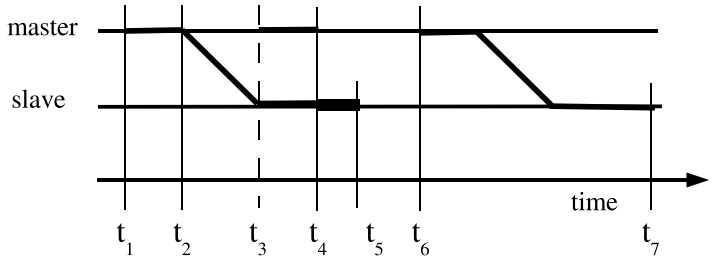


Figure 2.9: Improved 802.11 Synchronization

The protocol operates as follows. Periodically, the master creates a timestamp and broadcasts it. All nodes which are in range of the master receive the message simultaneously and adjust their clocks by their offset from the transmitter. The time-critical path of this algorithm (Fig. 2.8) is the Physical time between the creation of the timestamp on the master t_1 and when the slaves adjust their local clocks t_4. It therefore includes the propagation delay.

An algorithm which utilises the tightness property of the medium is presented in [11]. As wireless networks are suitable for broadcasts, the theories about tight reception of messages presented in [16] hold true. The algorithm is also designed for 802.11 wireless networks operating in infrastucture mode.

The access point broadcasts a indication message in each round (Fig. 2.9). On receipt of the message t_3, each slave (and the master) creates a timestamp for the receipt. In the following round, the indication message which the access point broadcasts at t_6 contains the timestamp which was created for the receipt of its own broadcast in the previous round. By comparing its own timestamp with the access point's, each receiver can calculate its offset from the access point when the original timestamps were created.

Due to the inherent unreliability of wireless communication, some degree of fault tolerance

is desirable. This is provided by the access point including timestamps for its transmissions in previous rounds in each new synchronization message. This means that if the access point includes timestamps for n previous rounds in each transmission, a receiver can synchronize with the access point if it receives two synchronization messages no more than n rounds apart.

The space time diagram shown in Figure 2.9 shows that the time-critical path has been improved over the original 802.11 synchronization [6]. Due to the tightness of reception of the message across all receivers (including the transmitter), the time-critical path only includes the variance in processing time $t_4 - t_3$. This is a step forward in terms of precision, as it manages to completely remove the propagation delay from the synchronization precision achievable by the algorithm.

This work is extended in [12] to provide continuous clock adjustment. The merits of this are discussed further in Section 2.6.

While this solution provides a means of synchronizing in single hop wireless networks, it does not extend to multi-hop networks due to its reliance on a central, coordinating access point. In addition, the assumption is made that the access point is fault tolerant, and a number of backups are required to provide this facility. It would not be possible to provide these in an ad hoc environment.

The algorithms in [6, 11] described previously are only suitable for single hop wireless environments. This is because broadcasts made on the wireless medium are received only by one-hop neighbours of the transmitter. In order that multi-hop synchronization be achieved, some alternate approach is required.

In [14], Sourour et al propose a strategy for inter-vehicle clock synchronization. The paper addresses the problem of providing a slotted ALOHA scheme for inter-vehicle communication. A pair of communication channels are required by the protocol. One channel is used for data transfer in slots, while the other channel is devoted to synchronization transmissions.

Each vehicle transmits a set of pulses at fixed intervals. The synchronization strategy aims to make these periodic pulse transmissions synchronous. The synchronization channel is constantly observed by each node. Other vehicles broadcasts of the set of pulses are compared to the nodes own transmission time for its set. This is similar to the Time Distance Capture logic described in [4] (Fig. 2.7). Using this technique, the offset of other vehicles can be calculated based on when they make their transmissions. Due to the fact that all transmitters are attempting to transmit

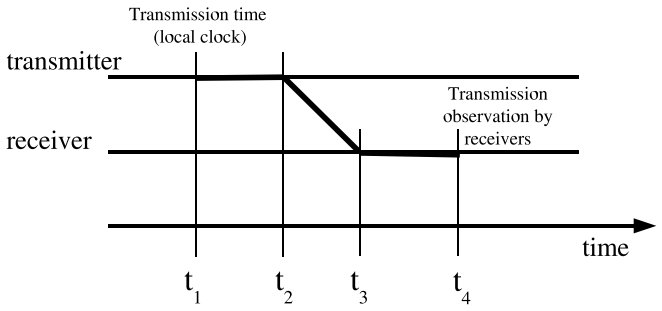


Figure 2.10: Sourour space time

synchronously, it is necessary to observe the power of received signals also. This is then used to take account of multiple vehicles transmitting synchronously, rather than treating the transmission as that of just one vehicle.

When a resynchronization point is reached, each offset observed is weighted based on the power associated with its transmission. These values are then averaged to calculate the offset which the resynchronizing node applies to its local clock. This algorithm deals with clock synchronization of multi-hop nodes through averaging. The algorithm doesn't attempt to read all the clocks in the network, rather only the one-hop clocks are read and synchronized to. The theory used is that if overlapping subsets of the total population are synchronized, then the global population will also be synchonized. The overlap between subsets acts as the middle ground which draws the average values in the subsets together. This synchronization can then be used to allocate slotted access to the data transmission channel.

This clock reading error is the same as that for [4] as it uses the same process to read a remote clock. Therefore the time-critical path includes the propagation delay and processing time (Fig. 2.10). While this algorithm was designed to deal with multi-hop ad hoc networks, the assumption that a node would have simultaneous access to a pair of transmission channels is restrictive. This protocol would therefore probably not be of practical use.

The two transmission channel issue is addressed in [1], which builds on the work of Sourour in [14]. The proposed solution uses predefined transmission slots similar to [4]. Again using the same process as Time Distance Capture logic (Fig. 2.7), a node's expected transmission time and actual transmission time can be compared to calculate the offset from a receiver's local clock. When a

resynchronization point is reached, the offsets observed are averaged and the result is applied to the local clock. Once again, local averaging is used to achieve global synchrony.

The time-critical path still includes the propagation delay as with Sourour's solution. While the requirement of defined slots is restrictive, it does mean that one-shot synchronization is still possible. This is very advantageous in the wireless environment where message loss can be high.

2.6 Local Clock Adjustment

As previously stated, there are two methods of adjusting the local clock - instantaneous and continuous correction.

Instantaneous correction means that the adjustment calculated is applied immediately to the local clock. This may result in jumps forward and back in the time on the local clock. An instantaneous forward correction will result in gaps in the local time, while a backward correction means negative interval measurements could occur [12].

Continuous correction spreads the clock correction over a period of time. Adjustments are made to the clock incrementally so it has changed by a specific amount at a defined point of time in the future. It has been shown that an instantaneous clock synchronization algorithm can be converted to a continuous one without loss in precision [3]. This would suggest that a continuous solution is preferable to an instantaneous one, as it improves the clock synchronization service provided without effecting it detrimentally.

2.7 Summary

Many clock synchronization approaches have been proposed. Analysis of these algorithm shows that their achievable precision is subject to three variables; clock reading error, physical clock drift, and variable processing times. Only the clock reading error can be addressed by a synchronization algorithm, as the other two parameters are properties of the physical environment and of the operating system in which the protocol is functioning.

Clock reading error has a bearing on how precision synchronization can be. The time-critical path is a measure of the clock reading error of an algorithm. The longer the path, the more uncertainty there is as to how 'well' a remote clock can be read. Different genres of clock synchronization

algorithm include the round-trip delay or propagation delay, with the latter reducing the clock reading error due to a shorter time-critical path. These algorithms can in turn provide more precise clock synchronization.

Solutions for wired networks are generally not suited to the wireless domain due to assumptions which they make about their environment. A number of clock synchronization approaches have been proposed for wireless ad hoc networks. A major advance is made in [11], which uses the tightness property of the medium to completely remove the propagation delay of messages from the time-critical path of the algorithm. While this algorithm is suitable only for a single hop environment, two algorithms [14, 1] use local averaging to achieve global synchrony.

Clocks can be adjusted either instantaneously or continuously. However continuous correction provides benefits without any loss in precision, and is therefore preferential.

Chapter 3

Solutions

Ideas selected from the algorithms described in Chapter 2 yielded three possible solutions to the problem of multi-hop clock synchronization. They all address the problem of multi-hop synchronization but with differing degrees of synchronization precision achievable.

3.1 Clock Drift

All three solutions presented are subject to the same error due to clock drift. Each solution relies on a pair of subsequent messages being received. Due to the unknown point at which a timestamp is created in a round, at worst two Π length rounds could pass before a timestamp is compared and used for synchronization. As discussed in Section 2.3, two clocks could therefore drift apart by at most $2(2\Pi)\sigma$ before a timestamp is used.

3.2 A Centralised Solution

This solution builds on the algorithm described by Mock in [11], which is a centralised solution suitable only for single-hop wireless networks. The algorithm can be extended to provide multi-hop clock synchronization using a simple idea from the Network Time Protocol [13]. NTP creates a dynamic hierarchical structure in which nodes that synchronize to a time server themselves become time servers of a lower rank.

With NTP two values are transmitted with each synchronization packet - its level in the hierarchy and the sum of the packet delay errors that occur at each hop between the primary server and its level. The packet delay error for each hop is a measure of the precision between the server and

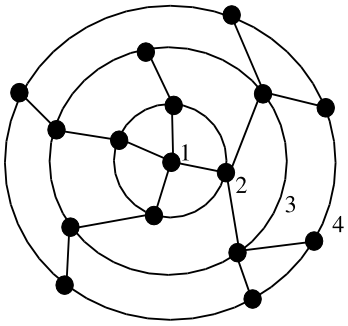


Figure 3.1: Layers of servers

client at that hop. Therefore she sum of packet delays is a measure of the synchronization with the primary server. These values allow a client to evaluate the source as a time server.

When a time server is chosen and synchronized to, the synchronizing node itself becomes a time server to other nodes, thus extending the hierarchy. When a client synchronizes to a server, it stores the level of that server and also the sum of the server's packet delays to the primary server. By incrementing the server's level by one, and adding its own packet delay to the server's total, it calculates the values which it transmits in its own synchronization packets for other clients to synchronize to.

This dynamic hierarchy could well be applied to Mock's protocol. In multi-hop ad hoc networks single-hop nodes would synchronize to the primary time server, become time servers themselves, and provide synchronization to two-hop nodes. The two-hop nodes become servers themselves and so on. This pattern continues until enough layers of time servers existed that all nodes in the network are synchronized (Fig. 3.1). Only the level at which the transmitter lies is needed in synchronization messages for a wireless protocol however, as the reading error will be the same for all nodes synchronizing to a server due to the tightness of the medium.

Unfortunately, it is not possible to use the procedure proposed by Mock in [11] to eliminate the propagation delay from the time-critical path. That solution timestamped an access point's receipt of its own transmission. However the network cards on ad hoc nodes are incapable of receiving their own transmissions, therefore that method cannot be used here.

Synchronization proceeds as follows. The primary node, to which all other nodes synchronize, broadcasts one synchronization message per fixed length period Π. The message is identified as $m(i_t)$, where i is the message identifier and t is the transmitting node identifier (Fig. 3.2). This

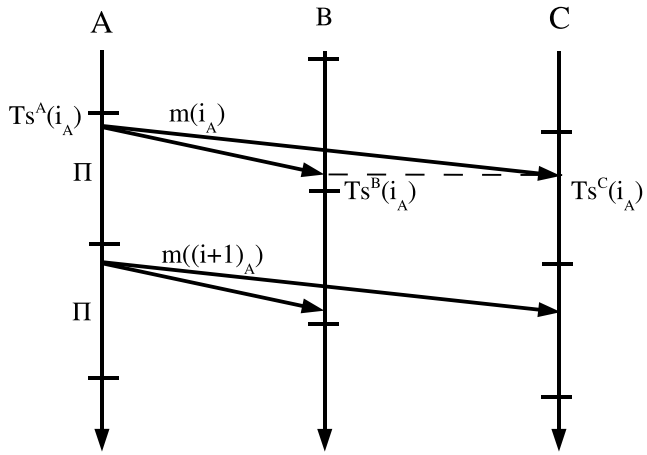


Figure 3.2: Message timestamping on transmitter and receivers

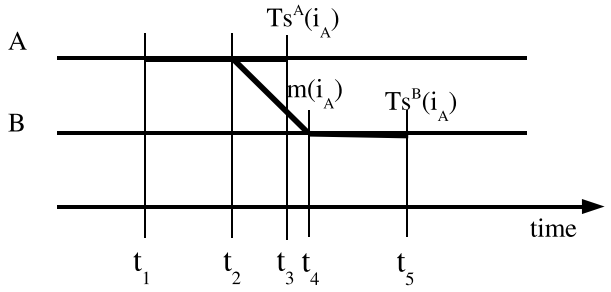


Figure 3.3: Extended Client-Server Space Time

allows each message from a transmitter to be uniquely identified. This transmission is timestamped on the server after transmission, and timestamped on each receiver upon receipt. Timestamps are of the form $Ts^r(i_t)$, where r is the receiving node's identifier, i is the message number and t is the transmitting node's identifier. In the subsequent period, the synchronization message from the server includes its timestamp for the transmission in the previous round. Upon resynchronization at the end of the round, each receiver adjusts its local clock by the difference between its timestamp for the transmission and the transmitter's i.e. it synchronizes to the transmitter. Each synchronized receiver increments the strata of the server it synchronized to and broadcasts a synchronization message in the next round identifying which layer in the hierarchy it resides on. This node is now a source of synchronization to other nodes.

The time-critical path of this algorithm (Fig. 3.3) contains the unknown propagation delay

and processing time. In addition the error due to clock drift (Section 3.1) affects the precision of synchronization achievable.

In order to calculate the synchronization precision achievable in the entire network, it is necessary to view the network as a series of single-hops. This means that the precision achievable between any node and the primary time server is the sum of the precision at each hop separating them.

There are a number of applicability issues with this protocol. One issue is that there has to be some means by which the central node of the system is chosen. The synchronization achievable in the network will be a function of the maximum number of hops from the primary server. Therefore a primary node selection algorithm should seek to minimize this value. Identifying this node is not a trivial problem due to the changing topology of an ad hoc network.

From a fault tolerance viewpoint, the primary node must also be monitored to check its correctness. It may not fail silently, and values broadcast could be incorrect. It might be possible to use a solution similar to that described in [16]. Each node is capable of starting a tentative clock on its neighbours, with one of these being chosen to be the new clock for the next round later. The subset of nodes in communication with the elected primary server could use this method to choose a node from one of their number to be the primary server for the next round i.e. use that node's tentative clock. This clock could be selected based on it being the median clock of those started. Statistically this makes that clock the likeliest to be correct.

How would this protocol handle the primary node being partitioned from the rest of the network? It is not immediately obvious how such a protocol would deal with partitions. If the network starts in a partitioned state, the nodes will elect a primary server on each side of the partition. This would have to be reconciled when the two partitions come together again. In order that the presence of more than one primary server be identifiable, each branch of the hierarchy would have to include in its synchronization broadcast a unique identifier for the primary server at the top of its tree. Nodes which receive messages with different primary server identifiers would have to initiate a protocol to resolve the network to a state where there is only one primary server.

If a partition occurs in the network after the nodes have already elected a primary server, then there will be no primary server on one side of the partition and another election must occur to elect a new primary server. When the partitions are merged again, the same problem will arise as discussed above, and there will be multiple primary servers transmitting to the same network.

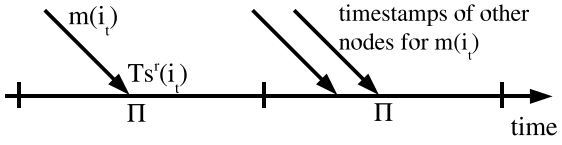


Figure 3.4: Resynchronization occurs between timestamp creation and forwarding

3.2.1 Resynchronizations between timestamp Creation and Comparison

An issue exists with the precision of timestamps when they are compared to calculate offset. Due to resynchronization taking place at the end of each round, the local clock may be adjusted in the period between a timestamps creation and it being forwarded in the following round (Fig. 3.4). The value of the timestamp is no longer consistent with current clock's view of that event's local time. If the clock after an adjustment is regarded as a new clock, this can be more simply understood as the old clock's value in the timestamp. This can be remedied by updating timestamps prior to forwarding them with value of the new clock i.e. apply any clock adjustment which is applied to the local clock to the timestamps also.

3.3 Two Distributed Solutions

The algorithms described in [14] and [1] both work on the basis that local synchronization of overlapping subsets of clocks will lead to global synchronization. This is also the basis on which the following pair of proposed algorithms operate.

The reliance of Sourour's algorithm [14] on a pair of independent transmission channels is a major stumbling point. Wireless network cards are capable of transmitting on only one channel at a time, therefore an implementation is required which uses a separate network card for each channel. Using one network card and switching between channels to transmit a synchronization message would not be possible, as the solution requires that the node also listens to that channel for the transmissions of synchronization messages from other nodes. These observations are then used to calculate remote clock offsets, and adjust the local clock.

It might seem possible to use the data channel for synchronization message. All nodes could transmit their synchronization messages at the start of a round, on the same channel as data is sent on. However, a side effect of all nodes attempting to transmit the synchronization message simulta-

neously, is that the messages will become corrupted when nodes actually become synchronized due to interference. While this will not matter to the synchronized nodes, unsynchronized nodes will not be able to observe how many nodes are transmitting at that point. It would also be impossible to tell the difference between corrupted synchronization messages and corrupted data messages. This solution is therefore unusable.

3.3.1 Slots for Synchronization

Due to the unlikely situation of all nodes having two wireless network cards, it is more probable that the solution described by Ebner [1] would be of use. Here each node is assigned a transmission slot, so that its transmissions do not interfere with the transmissions of other nodes. The slot transmission times must be well known, so that other nodes can observe the time between the expected and actual arrival time of messages. In addition, each message must include the identifier of the transmitter, so that it's slot can be identified and it's offset calculated. This solution works around the requirement that a pair of channels be available.

However the very nature of an ad hoc network is to be dynamic. Anything which is predefined is anathema to its dynamic nature. Therefore a valid goal of a clock synchronization algorithm for ad hoc networks is to minimise the prerequisites of the algorithm. Pursuit of this goal led to the notion of predefined slots being avoided, in favour of the approach adopted by the algorithms presented in the following two sections.

3.3.2 Tranmitter - Receiver Synchronization

This algorithm uses a method similar to that outlined in 3.2 to read remote clocks. Resynchronization occurs at Π second intervals. No requirement is made that nodes agree on the Physical time of these resynchronizations, merely that the length of Π is agreed upon. Clock adjustment is based on the offsets observed with one hop neighbours of the resynchronizing node.

Each node transmits a message $m(i_t)$ at a random point in each round, where i is the message identifier and t is the transmitting node identifier (Fig. 3.2). This allows each message from a transmitter to be uniquely identified. Due to the predetermined round length Π, a node will receive a message from its one-hop neighbours in each round, if no message loss occurs.

Timestamps are of the form $Ts^r(i_t)$, where r is the receiving node's identifier, i is the message number and t is the transmitting node's identifier. The timestamp of a specific transmission by a

Algorithm 3 1-hop Synchronization by Averaging

$$C_i(t) = \sum_{j \in S_i(t-1)} \frac{C_j(t-1)}{\#S_i(t-1)}$$

$S_i(t)$ is the set of clocks with which node i synchronizes in round t
$C_i(t)$ is the value of node i's clock for round t

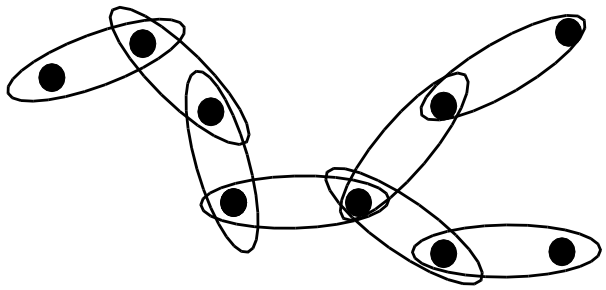


Figure 3.5: Subsets containing at least two nodes link the network together

node can therefore be identified. Timestamps are created for the transmission and receipt of each message. Upon transmission of a message, the transmitter creates a timestamp. In this case, the transmitter considers itself the receiver also of the message, so the value of r and t will be the same (Fig. 3.2). On receipt of the message, each receiver also timestamps the event.

The synchronization message transmitted includes the timestamp created for that nodes transmission in the previous round. A receiver can then compare its timestamp for the message to that of the transmitter. The difference between these timestamps is the offset of their local clocks at the point in Physical time at which the timestamps were created.

The same issue arises as is observed in the centralised solution (Section 3.2.1). A resynchronization point exists between the creation of the timestamp on the transmitter or receiver, and it being forwarded to neighbours for comparison. As before, this is simply solved by updating the timestamp's value with any clock adjustment which has occured since its creation. The timestamp then represents the latest view of the local clock's value.

This algorithm uses averaging to solve the problem of multi-hop synchronization (Alg. 3). In order for averaging to synchronize the whole population of nodes in the network, the nodes must be linked, via overlapping subsets, to each other. Local synchronization takes place inside each subset, and for this reason, there must be at least two nodes in each subset (Fig. 3.5).

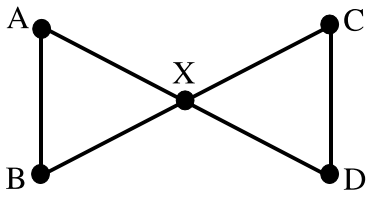
The reason local averaging achieves global synchronization is due to a node's clock value being

Algorithm 4 Multi-hop Synchronization

$$C_i(t) = function(C(t-h))$$

$C_i(t)$ is the clock of node i for round t
h is the number of hops separating node i and node j
$function$ means *is-dependent-on*

dependent on the values of it's one-hop neighbours clocks for the previous round. These values are in turn dependent on the original node's two-hop neighbour's values two rounds previous and so on. An example will illustrate this point.



If we look at the two-hop network pictured, the effect of multi-hop nodes on synchronization can be evaluated.

- $C_A(t) = \frac{C_A(t-1)+C_B(t-1)+C_X(t-1)}{3}$
- So $C_A(t)$is dependent on $C_X(t-1)$
- $C_X(t-1) = \frac{C_A(t-2)+C_B(t-2)+C_X(t-2)+C_C(t-2)+C_D(t-2)}{5}$
- Clearly then, $C_A(t)$is dependent on $C_X(t-1)$ is dependent on $C_C(t-2)$, $C_D(t-2)$
- Therefore (by Modus Ponens) $C_A(t)$ is dependent on $C_C(t-2)$, $C_D(t-2)$

This is generalised in Algorithm 4.

The time-critical path of this algorithm (Fig. 3.6) shows the inclusion of the propagation delay $(t_3 - t_2)$. This is because the receiver is comparing its timestamp for a message to the transmitter's timestamp. Again the error due to clock drift (Section 3.1) affects the precision of synchronization achievable.

The only assumption which this algorithm makes is that all nodes are aware of the round length. This reduces the prerequisites of the algorithm to below those required by the algorithms in [14, 1]. However it does not improve on the precision of these algorithms. This goal is addressed in the next algorithm proposed.

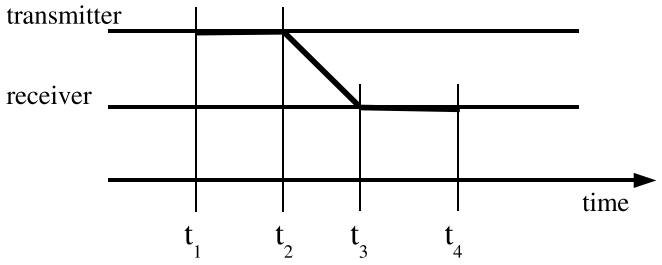


Figure 3.6: Transmitter-Receiver Synchronization

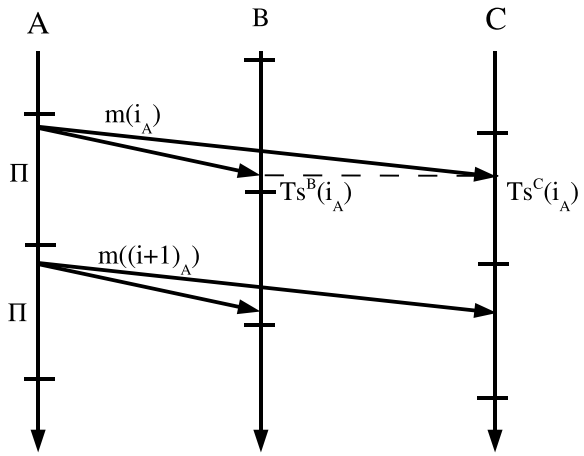


Figure 3.7: Message timestamping on Receivers only

3.3.3 Receiver - Receiver synchronization

This algorithm attempts to fully exploit the tightness property of the wireless medium as was achieved by Mock in [11]. This would remove the propagation delay completely from the time-critical path. However the ad hoc nodes which are synchronizing are unable to receive their own transmissions, which was the method by which Mock exploited tightness. Therefore an novel method of achieving this goal was required.

The process followed for this algorithm is the same as that described in 3.3.2, with one exception. When timestamps are being created, they are created only on receivers of messages, and not on transmitters (Fig. 3.7). These timestamps are them exchanged in the following round for comparison and offset calculation. Only receivers can exchange timestamps, and this places an extra requirement on the protocol.

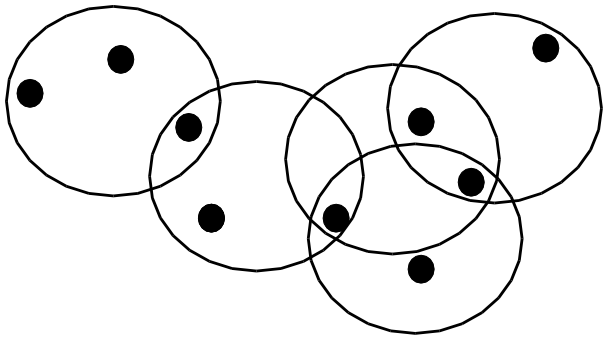


Figure 3.8: Subsets containing at least three nodes link the network together

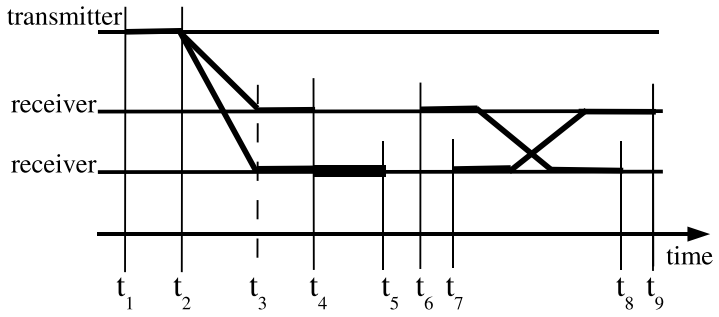


Figure 3.9: Receiver-Receiver time space

As with the previous algorithm (Section 3.3.2), this algorithm relies on local averaging to achieve global synchronization. The network must be linked by overlapping subsets in order that the global population can become synchronized. For local synchronization to occur, there need to be two receivers in transmission range of each other and a transmitter. This provisor permits receivers to compare their timestamps for the (tight) reception of the message from the transmitter. Basically there is a requirement that subsets consist of three or more nodes (Fig. 3.8).

By taking complete advantage of the tightness of the medium, the propagation delay is removed completely from this algorithm's time-critical path3.3.3. It contains only the variance $t_5 - t_4$. Once again the error due to clock drift (Section 3.1) affects the precision of synchronization achievable.

The assumption is made that the subsets consist of three or more nodes. In addition the round-length must be predetermined.

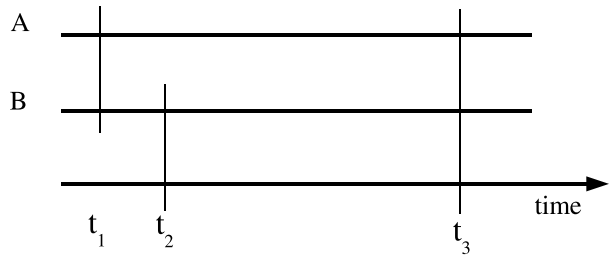


Figure 3.10: Coordination of synchronization points

3.3.4 The effect of Overlapping Slots

Due to there being no coordination of the physical instant at which synchronization occurs, separate nodes' concepts of which slot they are currently in may be inconsistent. As a result of this, a node may not receive it's neighbour's timestamp of a transmission until it has already used its timestamp for that transmission to resynchronize, and consequently discarded it.

In order to ensure that the physical time of resynchronizations align on separate nodes, continuous correction is applied to the local clock. This means that instead of the local time at which synchronizations occur being changed (as would happen with instantaneous correction), only the Physical time at which synchronization occurs changes. Therefore the local time separating the synchronizations will be consistent.

Due to resynchronizations occuring at fixed intervals from the start of the local clock, synchronizing the local clock therefore results in the synchronization in Physical time of these fixed intervals. An example can be observed in Figure 3.10.

Node A resynchronizes at Physical time t_1. Node B resynchronizes at Physical time t_2 . The goal is to make both A and B have their next resynchronization at t_3. If B adjusts its clock instantly to synchronize with A, then its next resynchronization will not be at t_3. However if B adjusts its local clock continuously, then its local clock value will show the next resynchronization point, but Physical time will only have advanced to t_3.

3.4 Summary

The centralised solution presented provides clock synchronization through a dynamic hierarchy of master/slave relationships. In this way, the entire population synchronizes through intermediaries

to a primary node which is selected randomly. The precision achievable between one-hop neighbours is limited by the propagation delay. The precision between the primary node and a node n hops away, is n times the precision between one-hop neighbours.

The two distributed solutions presented rely on averaging to solve the problem of global synchronization. Local synchronization of subsets which overlap results in the global population of nodes becoming synchronized. Local synchronization is achieved by the averaging of the clocks of a nodes one-hop neighbours.

The solution which uses synchronization between transmitters and receivers achieves a precision which is limited by the propagation delay. However, the solution for synchronization between receivers of common messages removes the propagation delay completely from the time-critical path.

All the proposed solutions provide clock synchronization for multi-hop ad hoc networks. While the solution in Section 3.3.3 has a further requirement over those in Sections 3.2 and 3.3.2, it does improve the precision achievable by removing the propagation delay from the time-critical path. Therefore this algorithm was the one chosen to carry on to the next stage of the dissertation.

Chapter 4

Design and Implementation

4.1 The Simulator

Following the selection of an algorithm, it was decided that the algorithms operation should be simulated as a proof of concept. It was hoped that observations of the model in operation would provide some understanding of how the protocol would behave; especially the behaviour of local averaging to achieve global synchrony. A mathematical solution predicting the upper bound on the precision of the algorithm has already been formulated (Alg. 3,4). However, it is still of some interest to observe how well the precision stays within the predicted upper bounds, especially in the presence of message loss. A simulator enables a numerical evaluation of the algorithm, and the gathered data allows the estimation of the characteristics of the model in various conditions [9]. Simulators can be continuous or discrete.

Few systems in practice are wholly discrete or continuous; but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous [9].

State variables change instantaneously at separated points in time in a discrete system, while they change continuously in a continuous system. The simulator designed was of the discrete system genre. This is not strictly true of an ad hoc network where the position, direction and speed of the nodes, as well as the local clock drift are changing continuously. However the main events which concern the system - broadcasts, receipts and synchronizations - occur at discrete times in the system. Therefore the choice to build a discrete, event-driven simulator was felt to be justified.

The system contains some random input such as processing time (see Section 2.2.2). Therefore

the simulator stochastic. *Stochastic models produce output which is itself random, and must therefore be treated as only an estimate of the true characteristics of the model* [9].

Discrete-event simulation involves stepping forward in time to events which effect the state of the system; in this case broadcasts, receipts and synchronizations. A simultation clock tracks the passage of simulated Physical time as it progresses. The system of time advancement is *next-event time advancement.* The simulation clock is initialised to zero, and the time of future events is determined. The simulation clock is then systematically advanced to the next event which occurs, at which point the state of the system is affected by that event's occurance. Future events are also added as a result of events which occur e.g. a broadcast has a response. The size of the steps in Physical time in the system are not consistent as they are dependent on when events occur.

The variables detailing the system state are output at regular intervals, providing data for analysis. These statistics include the offset of clocks from the local average, synchronization precision and synchronization accuracy. In order that different scenarios be analysed, the simulator provides a number of options on startup. Static nodes in specific topologies can be simulated, as well as randomly distributed nodes, in order that the operation of the protocol in a multi-hop environment can be evaluated.

4.2 Simulator Assumptions

In order to facilitate a discrete-event simulator, a number of assumptions had to be made.

- The direction and speed of the nodes in the network is assumed to be consistent between discrete events. The direction of the node for the interval between the previous event and the current event is chosen randomly. The speed of nodes in the network is defined prior to simulation startup. The distance travelled is calculated based on the time passed and the set speed of the node. The new position of the node is then calculated based on its random direction and the distance travelled.

- Physical clock drift is affected by environmental conditions such as temperature and humidity. Rather than simulating these conditions, a clocks drift rate was chosen randomly within a bound at the start of the simulation. This value was then taken to be the drift of that clock for the entire simulation.

4.3 Design

The flow of control inside the simulator is shown in Figure 4.1.

1. Upon initialisation, the simulator clock is set to zero. The ad hoc nodes which maintain the state of the system are created, and assigned random drift rates and positions on the "map". The first events which affect each node in the simulator are added to the list of events.

2. The simulation clock is advanced to the next event to occur.

3. The event is identified, and its effect updates the system state. Future events are added to the list as a result of this event. The new system state is output for analysis. The various new events which can be created based on the currently occuring event are:

   - Broadcast(node_ID, message) event results in a Receipt(message) event and the next Broadcast(node_ID) event.
   - Receipt(message) event creates no new events.
   - Synchronization(node_ID) event results in the next Synchronization(node_ID) event being added.

4. Repeat steps 2 and 3 until the exit condition is met.

The class diagram for the simulator is shown in Figure 4.2. It should be noted that there is no timestamp class. The value of the timestamp is stored inside the message upon its receipt.

4.4 Implementation

The simulator was written in the Java (Release 1.4.1_02) programming language. This was simply due to the author's familiarity with the language. A test-driven approach was adopted during development. The jUnit test suite was used to write the tests.

The simulator has a graphical user interface (GUI) which offers the user a menu on startup. The user can choose to simulate fixed topologies or randomly moving nodes. The maximum drift rate, processing time and other variables are configurable through property files.

The implementation uses the java.Math.random() function to create the stochastic environment of the simulator. On initialisation of the Simulator, the Ad hoc nodes which maintain the state

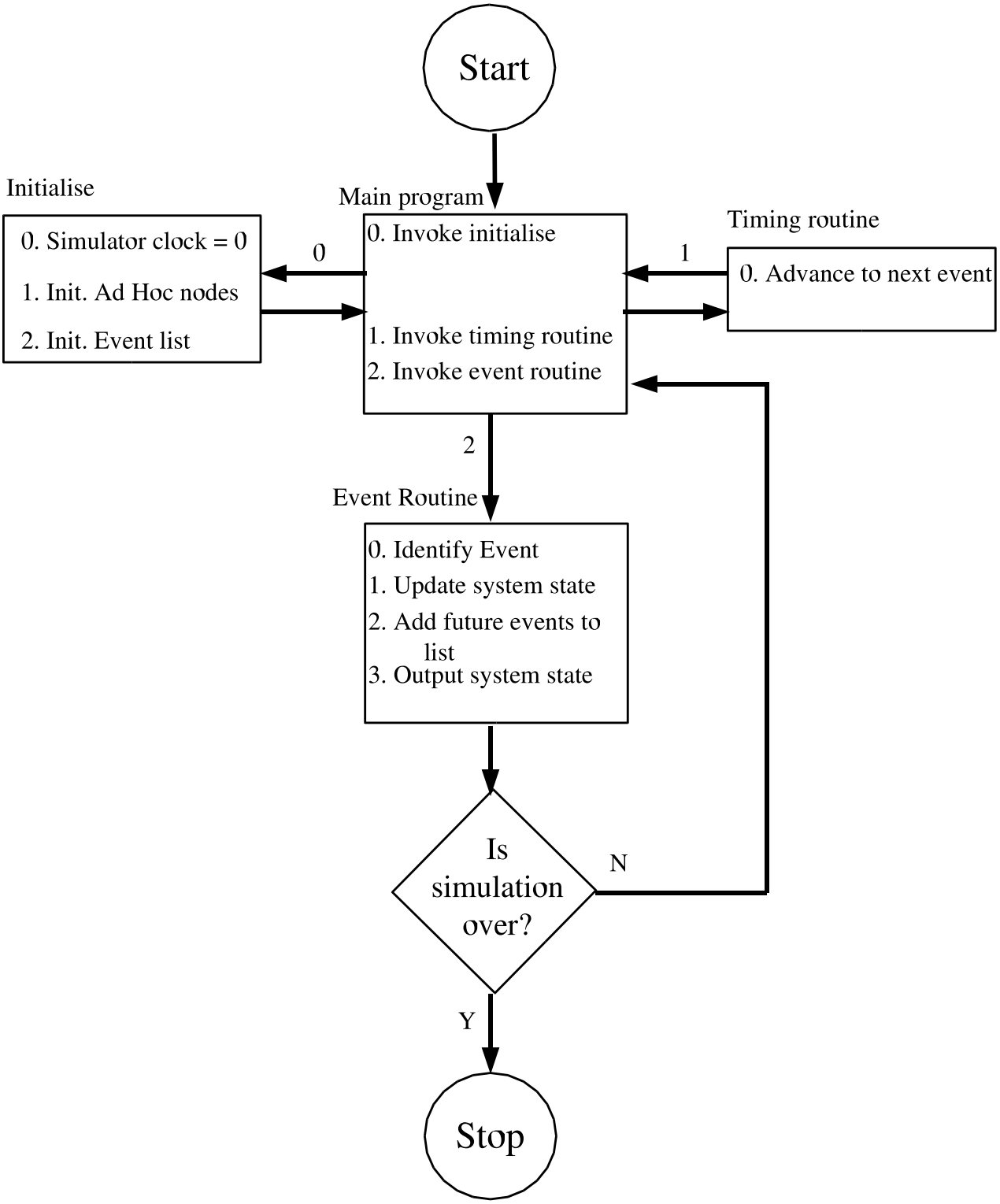


Figure 4.1: Flow of Control in the Simulator

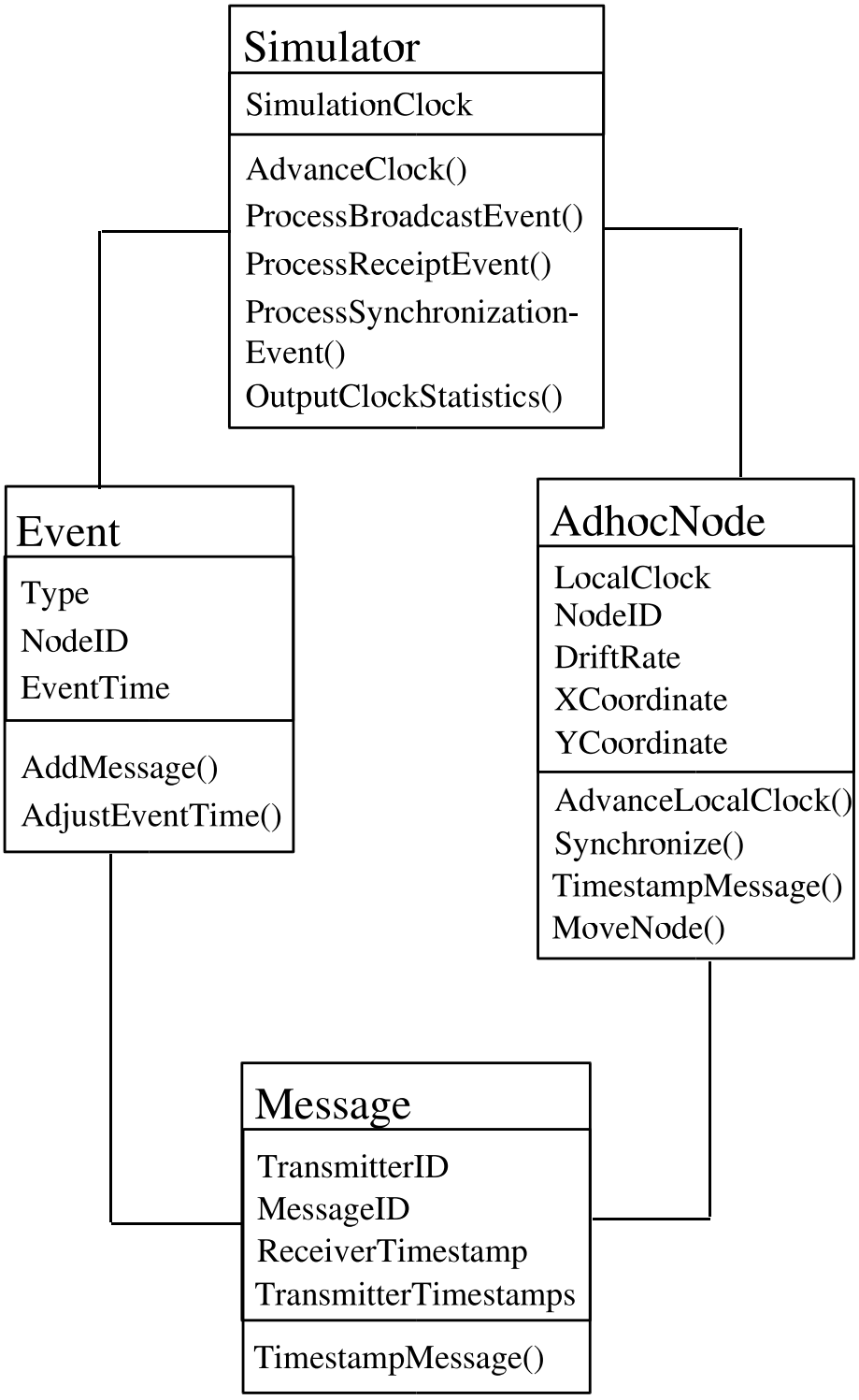


Figure 4.2: Class Diagram of the Simulator

Property	Description
MaxDriftRate	Maximum drift which a clock can have either positively or negatively (%)
SynchronizationInterval	Length of time between resynchronizations on a node (seconds)
ProcessingTime	Lower bound on processing time for creating a timestamp (seconds)
ProcessingTimeDifference	Maximum difference between upper bound on processing time and lower bound (%)
numberOfNodes	Number of nodes in the network
maxX	Maximum X coordinate of the map (metres)
maxY	Maximum Y coordinate of the map (metres)
transmissionRange	Maximum transmission range of the medium (metres)
clockStartInterval	Range of values in which local clocks can be initialised (seconds)
nodeSpeed	Fixed speed at which nodes travel in the network (metres/second)
nodeLayout	Static location of nodes (node speed must also be set to zero)

Table 4.1: Configurable variables of the simulator

of the system are created. These are assigned random values for local-clock-drift, start-clock-value and start-location within the bounds defined in Table 4.1.

The simulator then creates the first synchronization and broadcast events for each node in the network. The simulator clock times for these events are adjusted to allow for the clock drift i.e the time at which the event occurs is at the time which the event node expects it to occur at. The simulator then proceeds to step through the events as they occur.

- **Broadcast Event**

  When a Broadcast event occurs, a new Message is created using the transmitter_ID and its next message_ID. This message is filled with the timestamps which the transmitting node created for message receipts in its previous round. The message is then tagged onto a new Receipt event which is inserted into the Event List after the propagation delay defined in Table 4.1. A new Broadcast event is also added for the transmitting node's next broadcast.

- **Receipt Event**

  When a receipt event occurs, all nodes in transmission range of the transmitting node are identified. These nodes are individually evaluated based on their distance from the transmitter, and are assigned a packet drop probability. Based on this a message is delivered or not.

  A new timestamp is created on receivers for that message. This timestamp is created after

some random processing delay between the bounds defined in Table 4.1. The transmitters timestamps contained in the Message are used to update the receivers table of timestamps.

- **Synchronization Event**

  When a Synchronization event occurs, the synchronizing node calculates the offsets of remote clocks using its table of timestamps. These offsets are then averaged calculate the adjustment which should be applied. This adjustment is then applied to the local clock. A new Synchronization event is added. The timestamps which the synchronizing node created in the previous round are adjusted by the same clock adjustment to reflect the new clock.

4.5 Summary

The simulator is of the *discrete stochastic event* genre. This type was decide upon based on the major events in the system being discrete (broadcasts, receipts and synchronizations), and there being a degree of randomness in the system (processing time). This results in some assumptions being made about the continuous portions of the system (clock drift, node movement).

The simulator itself is built in Java, and uses property files to configure the system. Events are stepped through as they occur and the state variables of the system are affected by them.

Chapter 5

Evaluation

The simulator enables the examination of the protocol in various environments. The state of the model can be followed as simulated time progresses, to gauge the behaviour of the algorithm.

5.1 Topologies examined

The operation of the algorithm was evaluated a a number of static configurations of network topology and also where the nodes were free to move around, in a dynamic network topology. The static topologies were designed to simulate the presence of an increasing number of hops in the network. This enabled the examination of how well the synchronized clocks stayed within the bound which was mathematically formulated.

The operation of a network from one up to five hops in length (Fig. 5.1) was simulated and analysed. In addition, the operation of the protocol when the the network is partitioned was simulated. In this case, the network started with nodes divided on each side of a partition. The nodes then moves randomly until the partition was merged.

5.2 Packet Loss

Each topology was simulated with and without packet loss. The mathematical bound placed on the synchronization precision only holds true in the absence of message loss. Therefore it was desirable to observe how well the bound was maintained in the network when message loss occured. The two levels of packet loss used for each experiment were 0% loss and 45% loss.

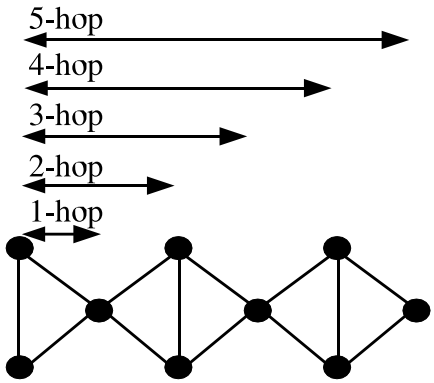


Figure 5.1: Multi-hop topologies simulated

5.3 Simulator Configuration

The variables which govern the operation of the simulator were configured for testing using the following values. See Table 4.1 for a description of the variables.

- **Ad hoc Node Variables**
  - MaxDriftRate = 0.000001
  - SynchronizationInterval = 10
  - ProcessingTime = 0.00001
  - ProcessingTimeDifference = 0.5
  - nodeSpeed = 0 (except in the case of the Partitioned network tests where nodeSpeed = 1)
- **Network Variables**
  - clockStartInterval = 100
  - propagationDelay = 0.001
  - maxTransmissionRange = 250
- **Topology Dependent Variables**
  - numberOfNodes

- maxX
- maxY
- nodeLayout

These values were used to calculate the theoretical upper bound on synchronization precision. The precision achieved in each simulated topology was then compared to this bound.

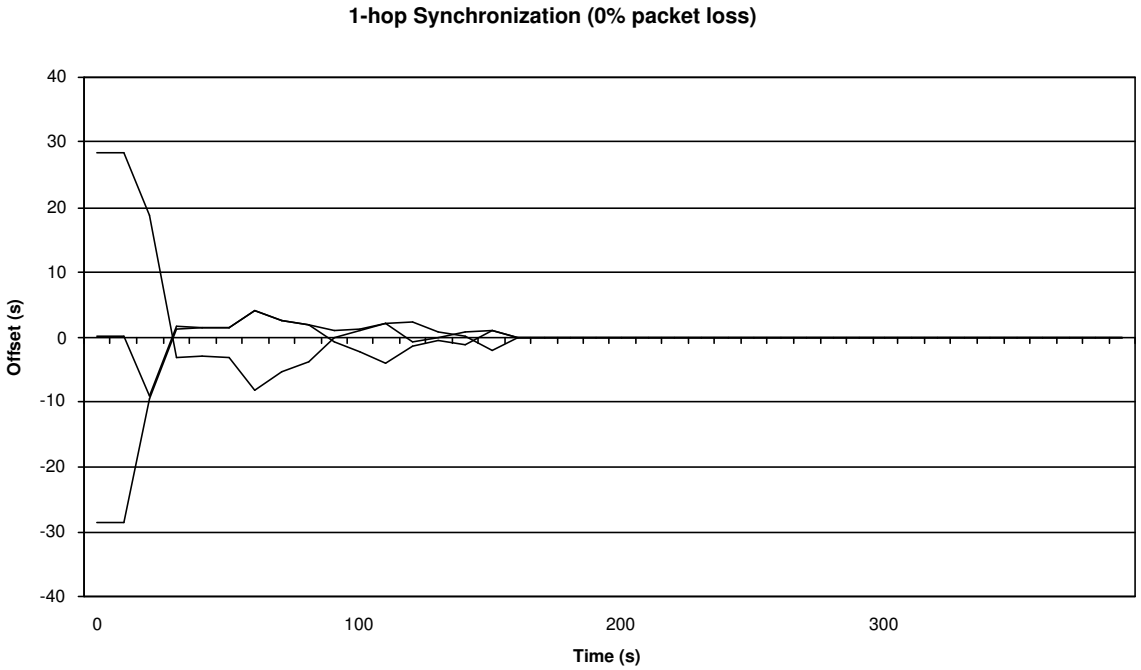
5.4 Synchronization

This section contains the graphs analysing the synchronization of different topologies until a steady state is reached. A steady state is where the precision of synchronization achieved has ceased to improve. This is discussed further in Section 5.5. For ease of comparison the graphs showing the path to synchronization of each topology, with and without message loss, have been grouped together. The time scale for pairs of graphs has been kept consistent, also for ease of comparison.

Figure 5.2 shows the graphs of a single hop network synchronization. It can be seen that there is a marked difference between the time taken to resynchronize with and without packet loss. It takes much longer to synchronize where packets are dropped. This trend can be observed to continue in Figures 5.3 - 5.6. Both message loss and increasing number of hops increase the time taken to achieve synchronization.

The partitioned network graphed in Figure 5.7 shows some interesting features of the protocol. The simulation where packets are lost is particularly interesting. The groups of nodes on each side of the partition achieve synchronization on their own side almost immediately. As the groups experience some overlap, the overlapping members are drawn towards the average of the two groups. These members are then drawn back into their own group as they cease to synchronize with both sides of the partition. This temporary overlap can be seen to cause spikes on the graph. Gradually the overlap draws the two sets together until eventually global synchronization is achieved.

While the time until synchronization seems very long, this can be attributed to the relatively long round length used in the simulation (10 seconds). It is better to evaluate the algorithms performance based on how many rounds it takes to achieve a steady state (Fig. 5.8). There is steeper incline on the line which reflects the results when packets are dropped. This reveals that packet loss has more of an effect than increasing rounds on the time required to achieve a steady state.



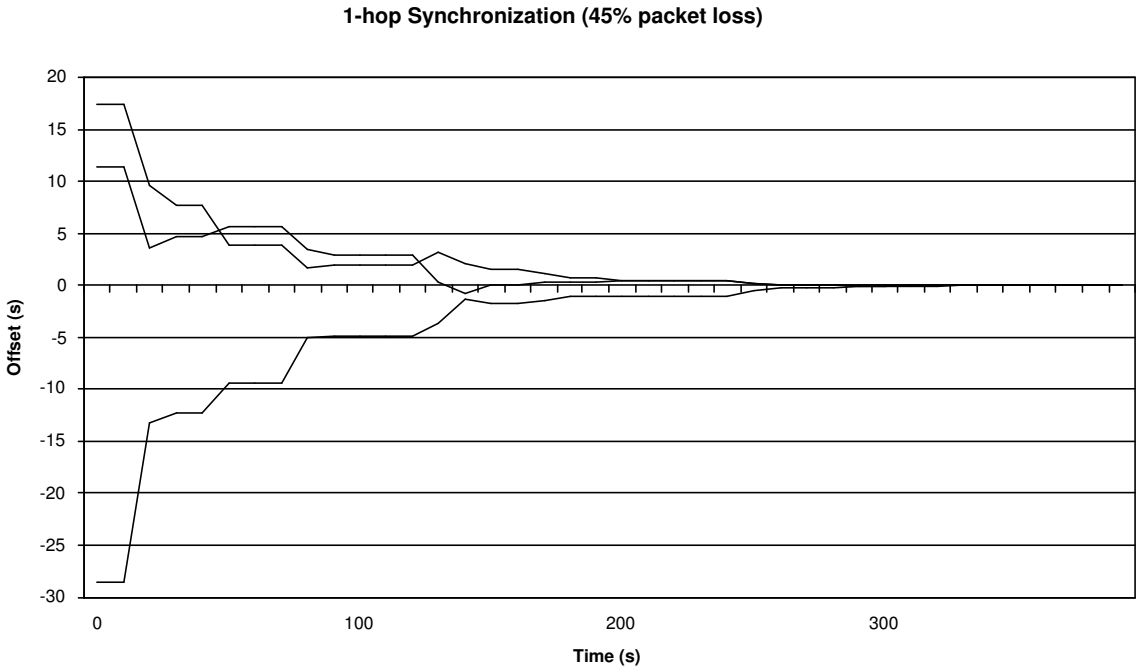


Figure 5.2: 1-hop synchronization

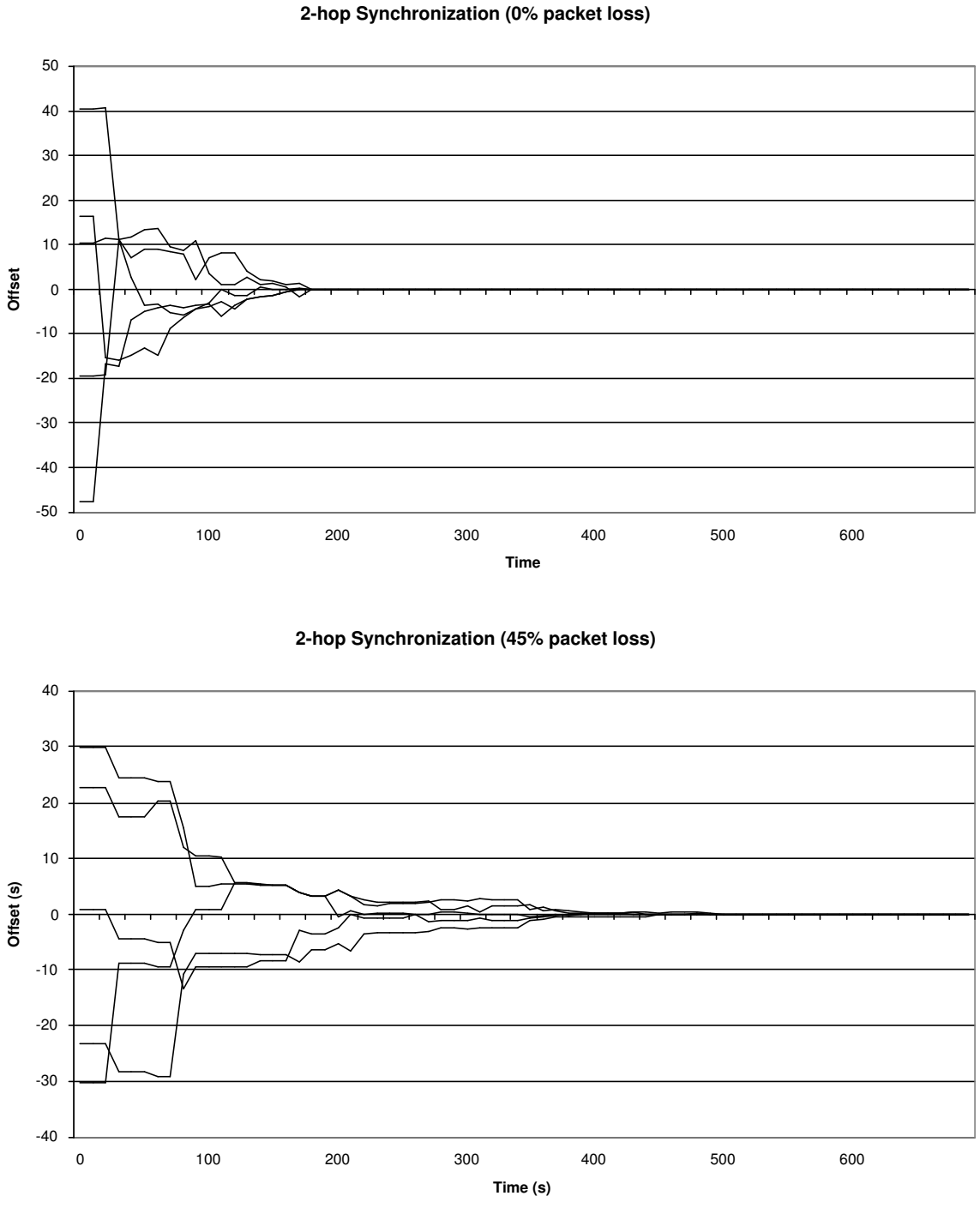


Figure 5.3: 2-hop synchronization

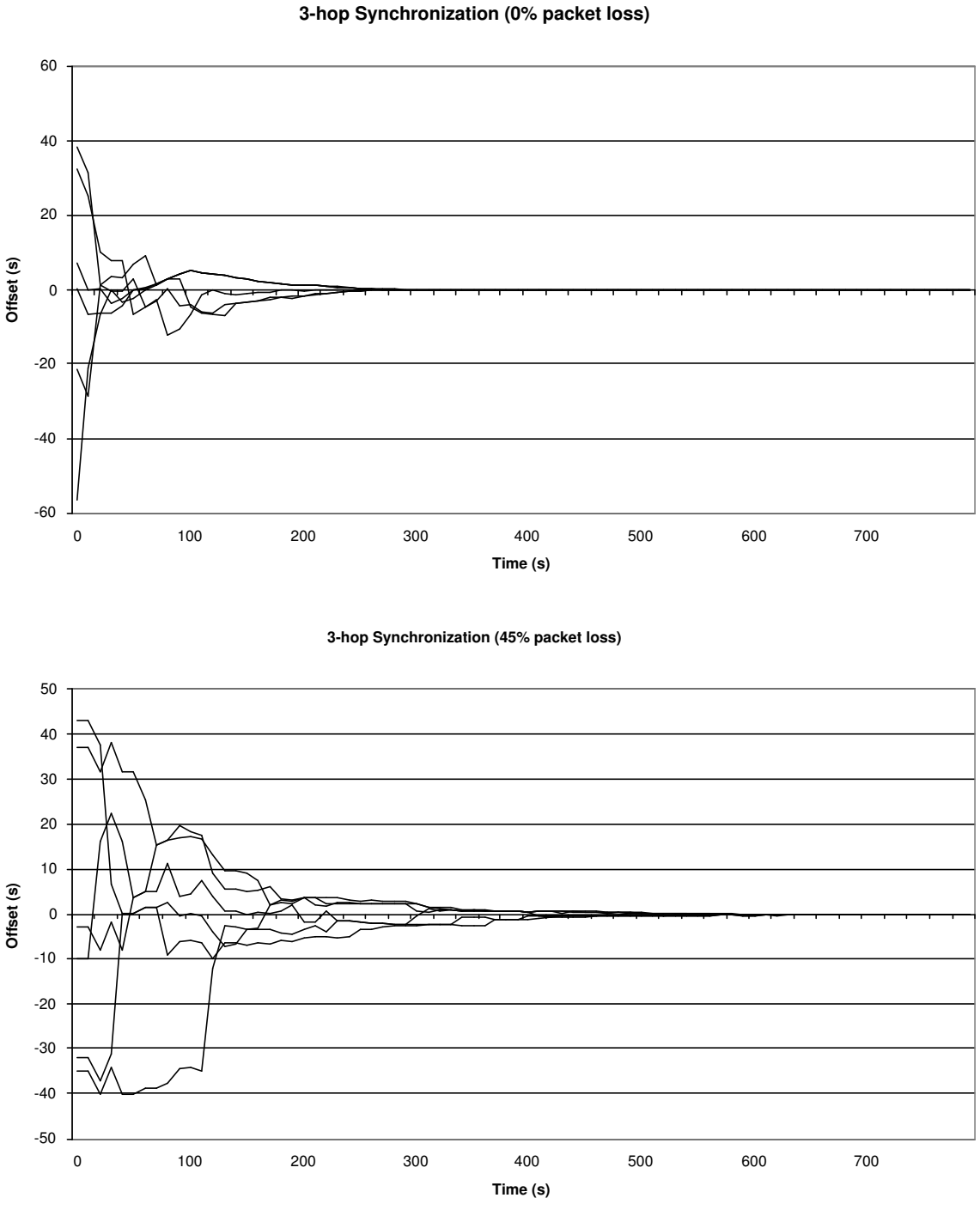
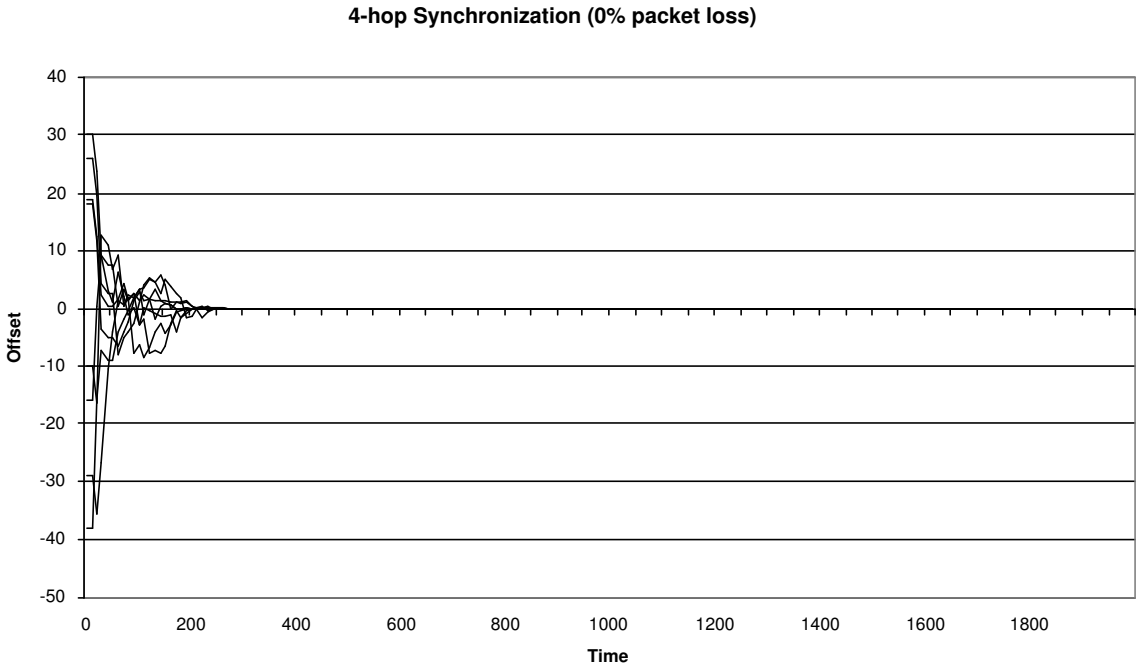


Figure 5.4: 3-hop synchronization



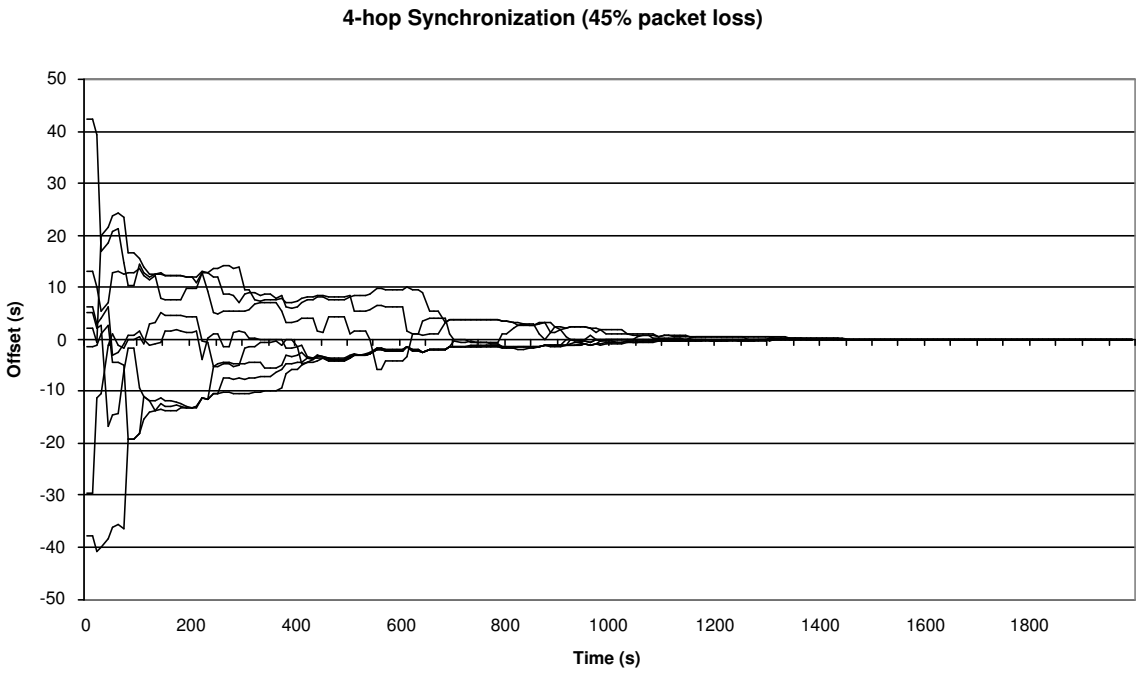
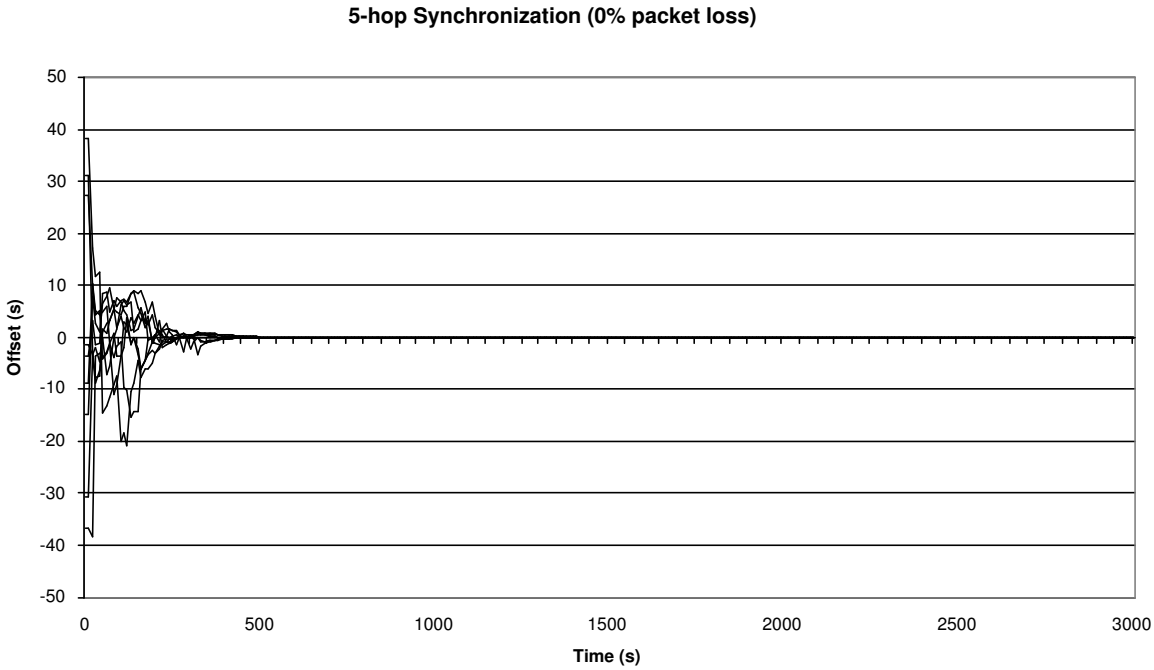


Figure 5.5: 4-hop synchronization



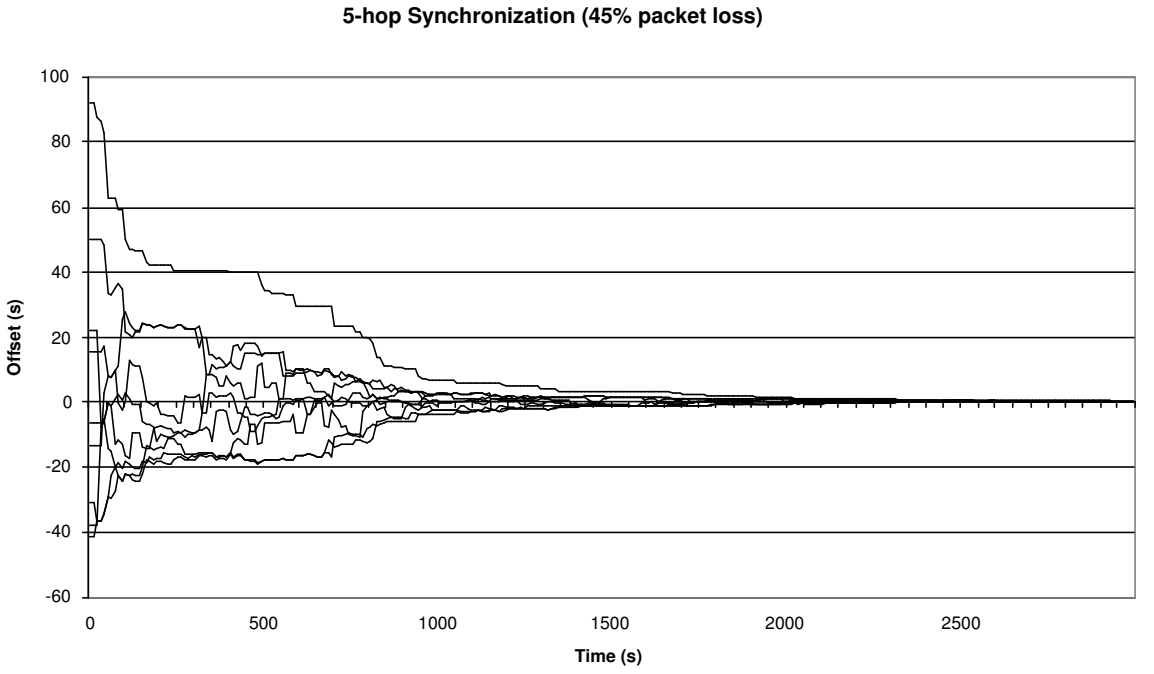
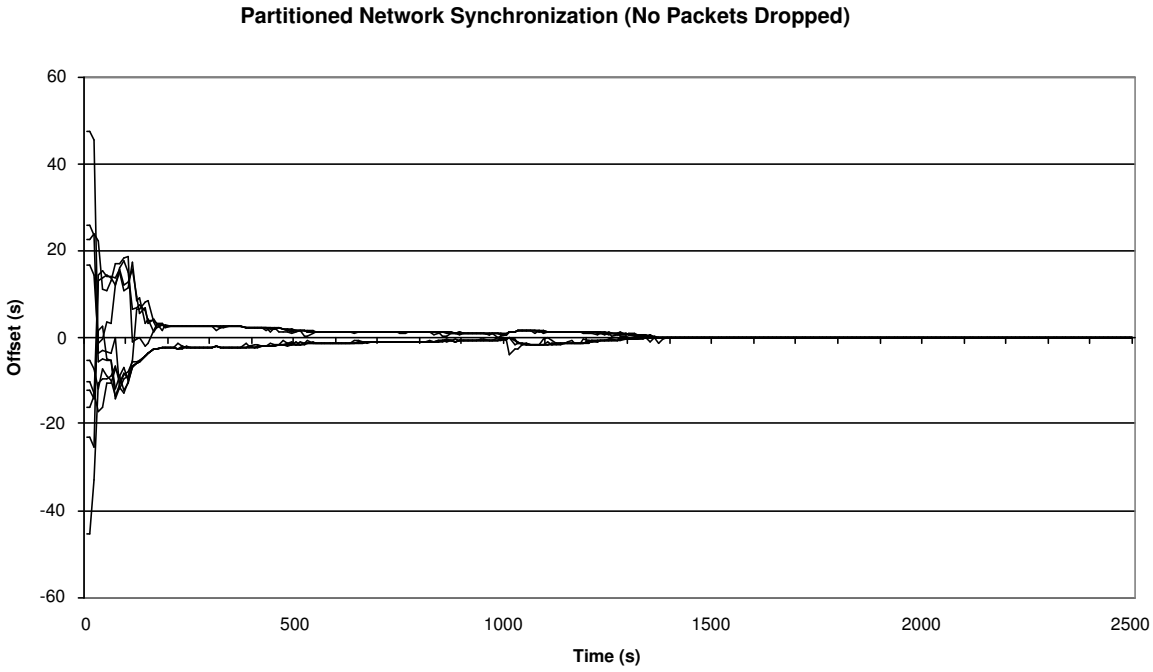


Figure 5.6: 5-hop synchronization



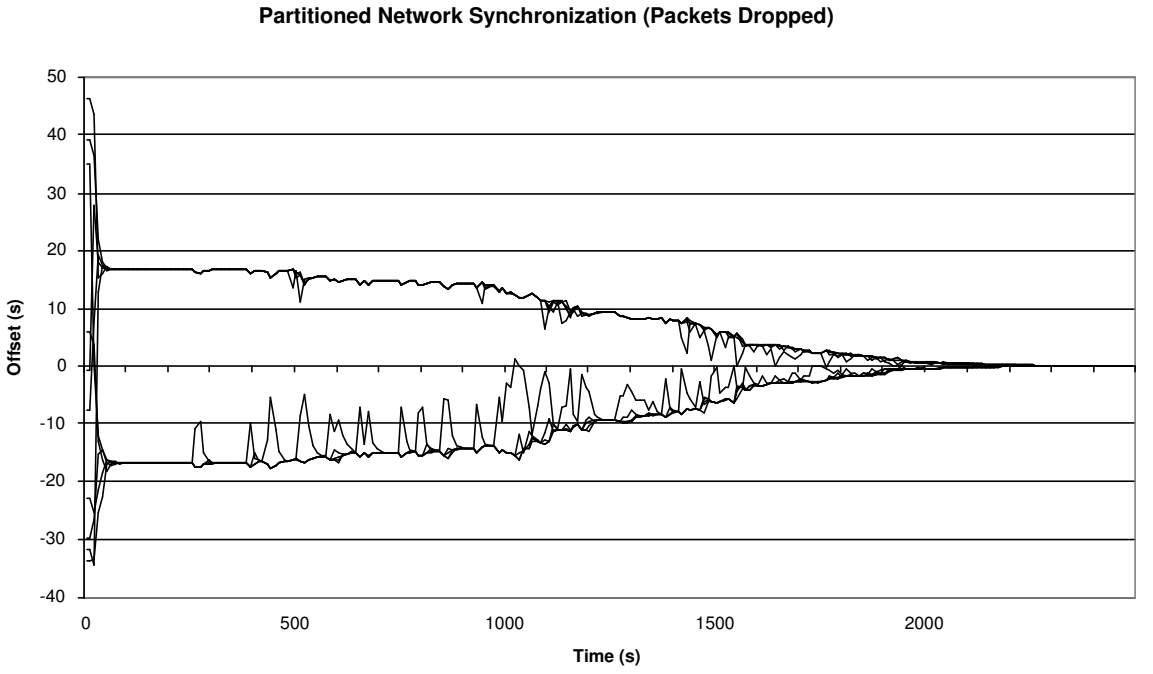


Figure 5.7: Partitioned Network at startup

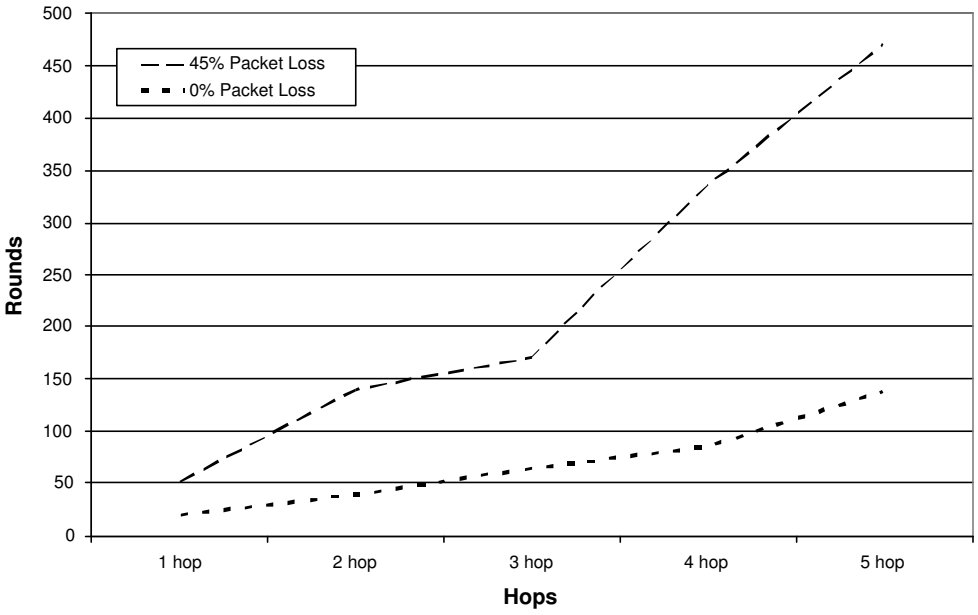


Figure 5.8: Rounds to reach Steady State

5.5 Precision Achieved

The precision achieved by the algorithm can be examined once a steady state has been reached. The steady state is the point in simulation at which the precision ceases to change considerably. Only oscillation due to varying processing time or missed resynchronizations (due to message loss) can be observed. The precision of the algorithm in different topologies and conditions can be compared

It is observed in Section 5.4 that increasing hops and message loss has a detrimental effect on the speed of synchronization. The same is true of precision. The precision achieved in a single hop network is graphed in Figure 5.9. This reveals a considerable difference in the precision achieved with and without packet loss. The oscillation in the graph of the simulation where packets were dropped also shows much greater variance in value.

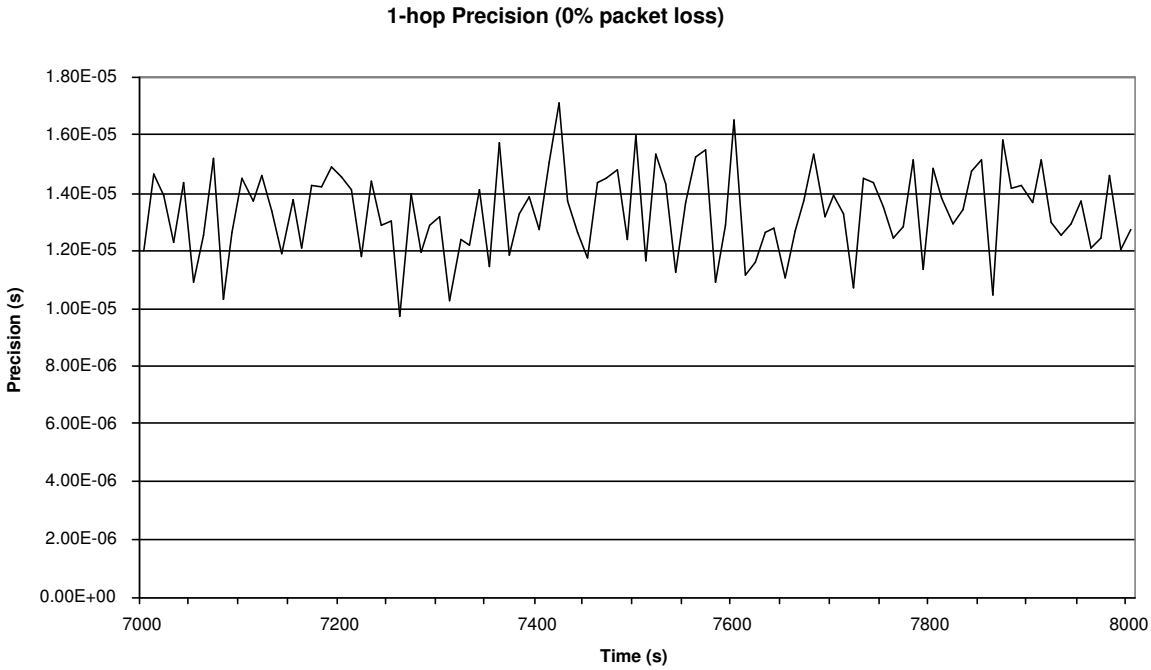
This trend is once again carried on in networks of increasing diameter (number of hops). Based on a value of 45% message loss, and the algorithm's requirement that two subsequent messages be received, it is straightforward probability theory to calculate that only 30% approx. of the possible resynchronizations will actually have values to compare.

The effect of message loss can be seen to be far more detrimental than increasing hops to the algorithm's precision in Figure 5.14. This graph shows the deterioration in performance as hops and message loss increase in the network. The line representing precision in the loss-less network is consistently well below the projected bound for that number of hops. In comparison, the line representing precision in the 45% message loss network is constantly verging on breaking the bound and eventually does so when the network diameter reaches five hops in length.

5.6 Accuracy

Although the aim of the algorithm is to achieve a higher degree of precision than other algorithms, it is still of interest to observe how closely the synchronized clocks are linked to Physical time i.e. how accurate the algorithm is. Figure 5.15 shows the accuracy of the average clock value in a single hop network as clocks synchronize. Upon initialisation the Physical time of the system is taken to be the average value of the set of clocks.

These clocks are initially spread across a one hundred second interval, and this is reflected in the dramatic changes of the graphs. The graphs with and without message loss do not reveal any further information by being compared, as they are reflective of different startup conditions due to



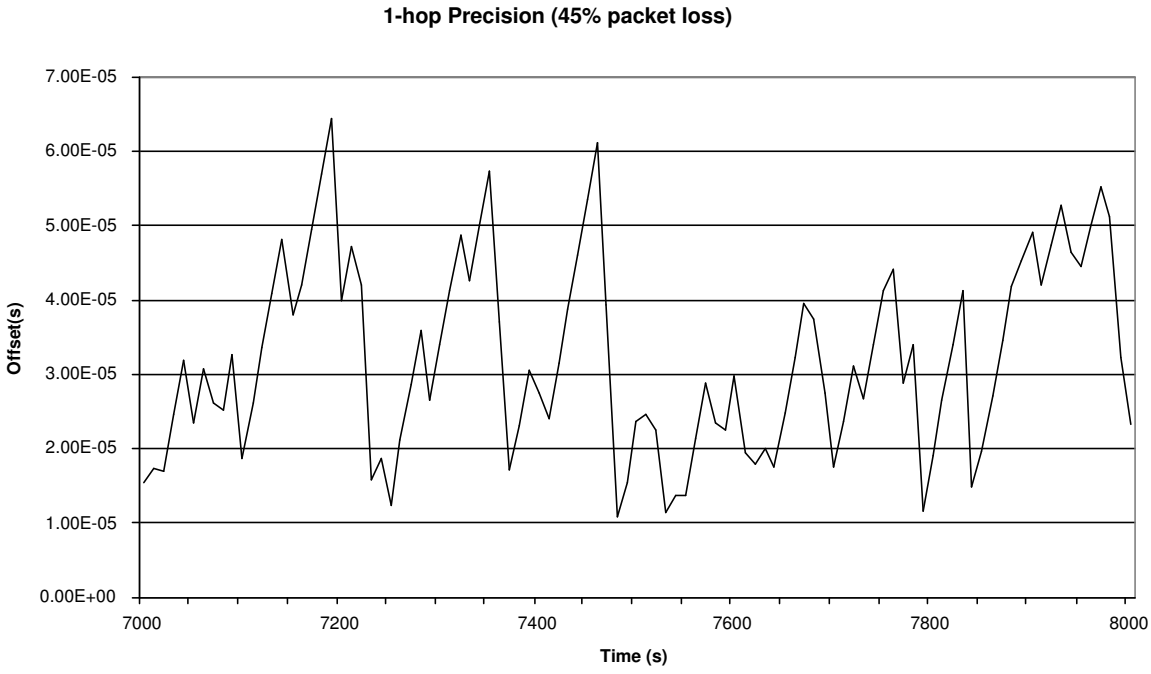
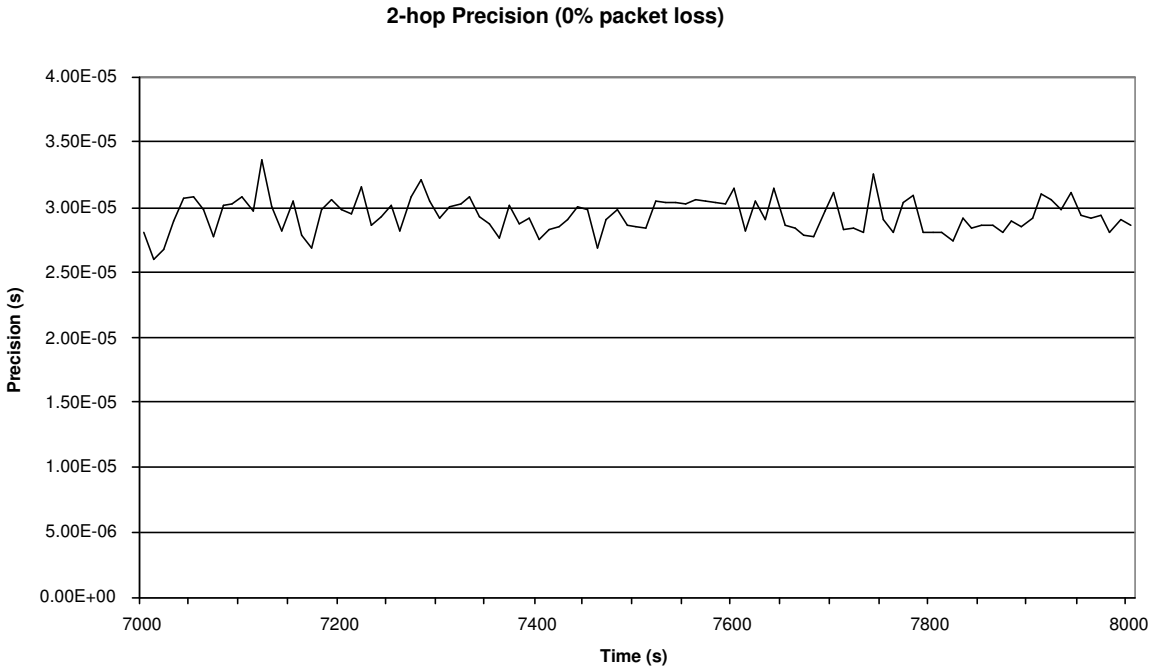


Figure 5.9: 1-hop precision



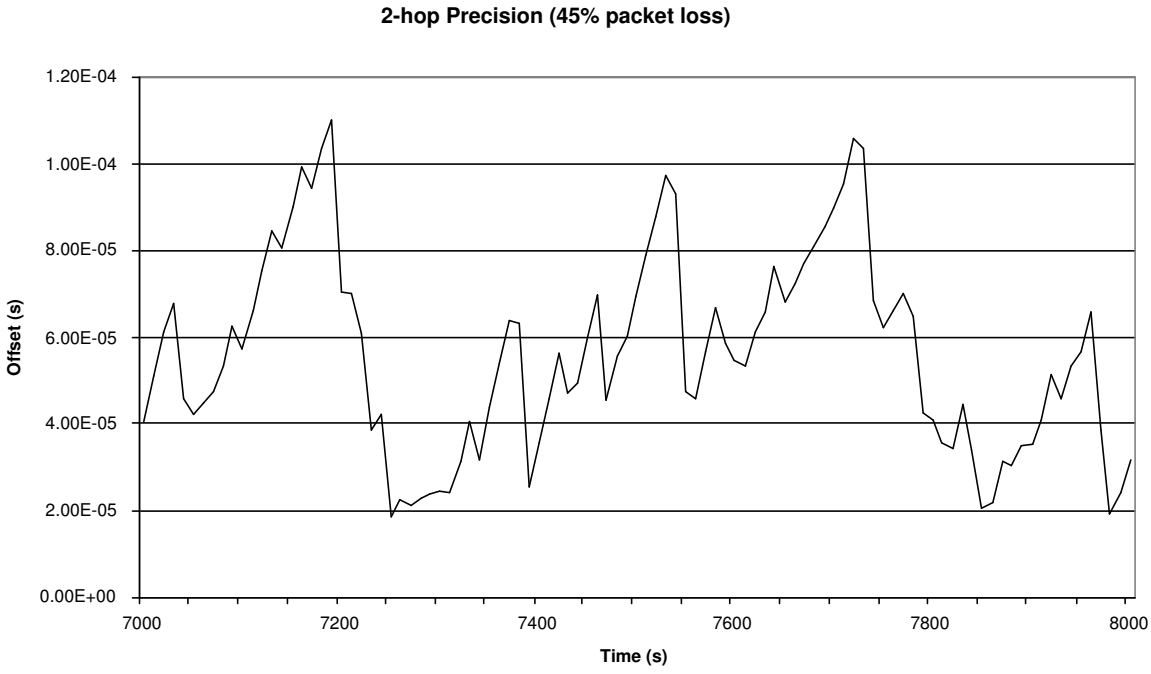
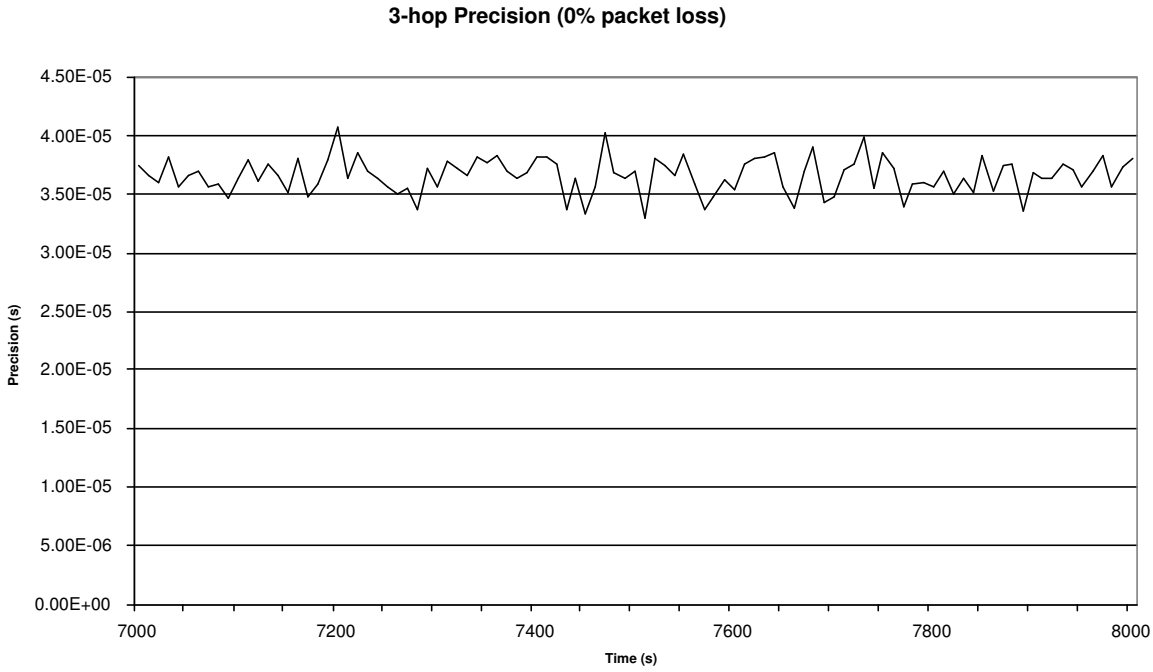


Figure 5.10: 2-hop precision



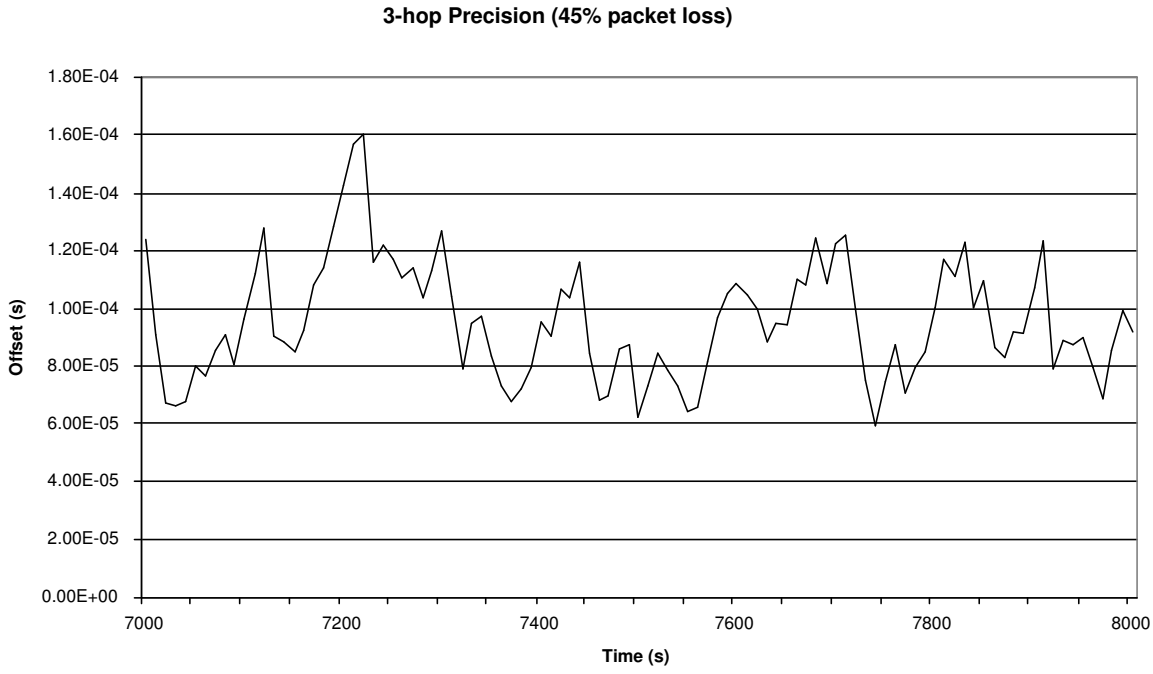
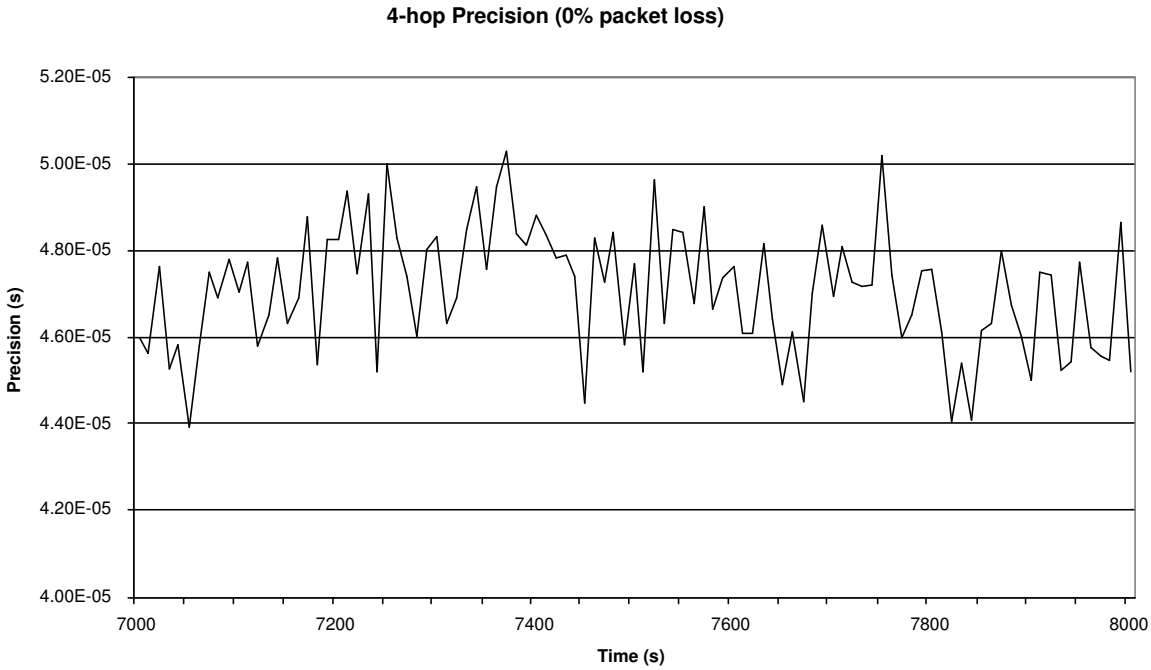


Figure 5.11: 3-hop precision



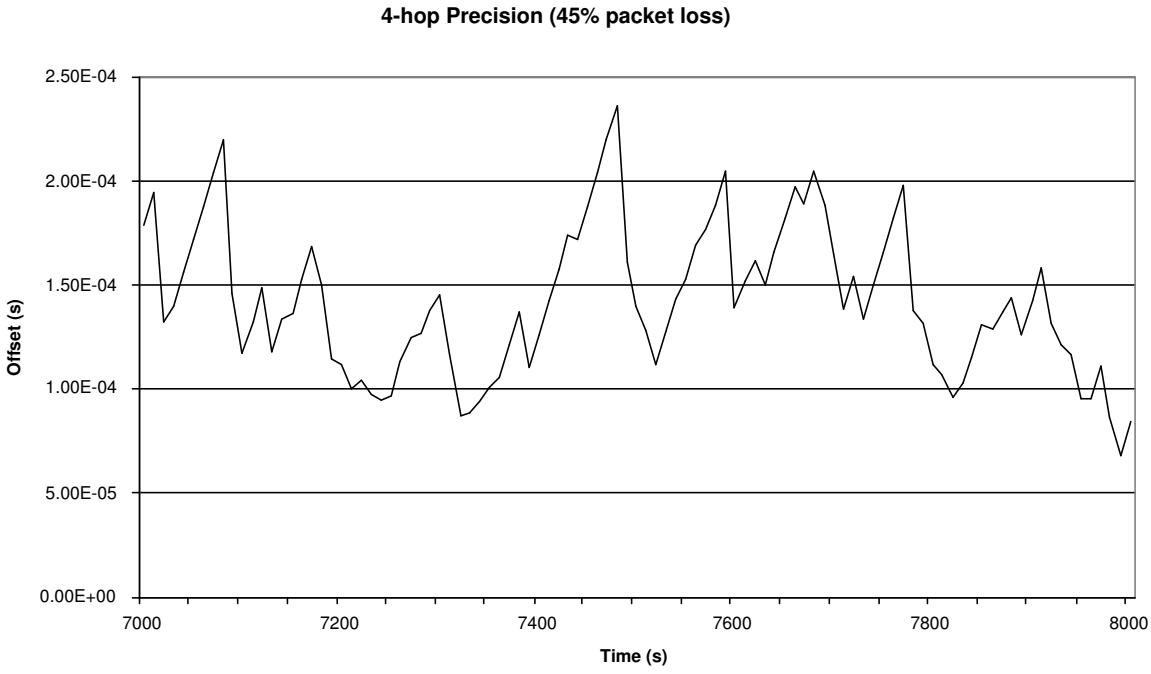
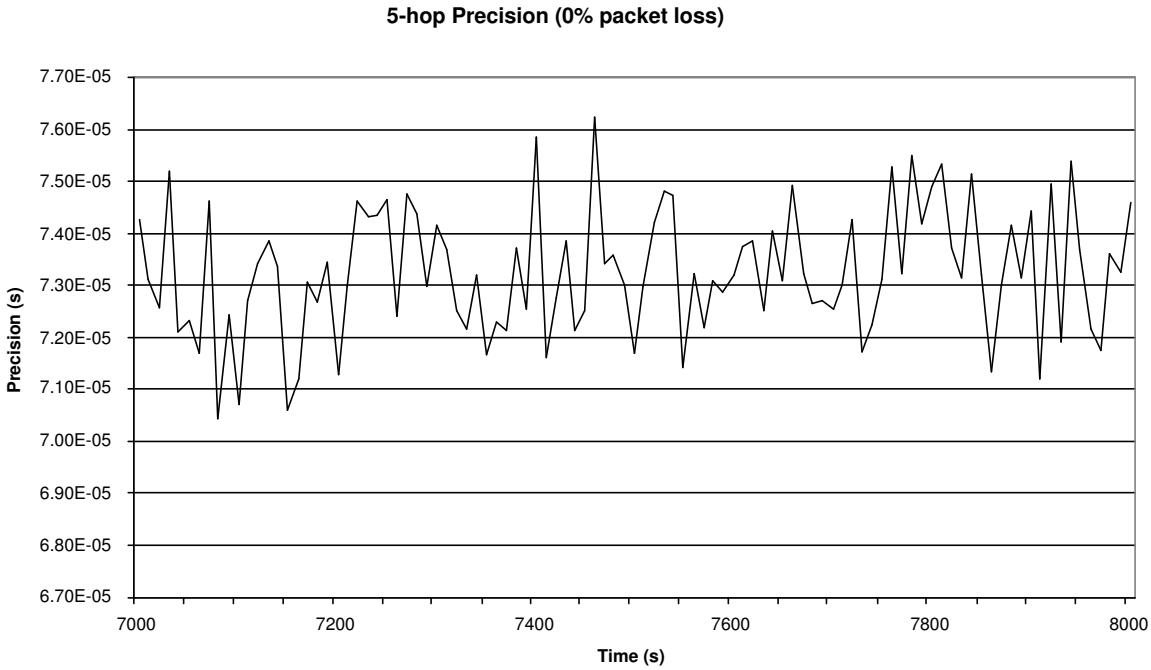


Figure 5.12: 4-hop precision



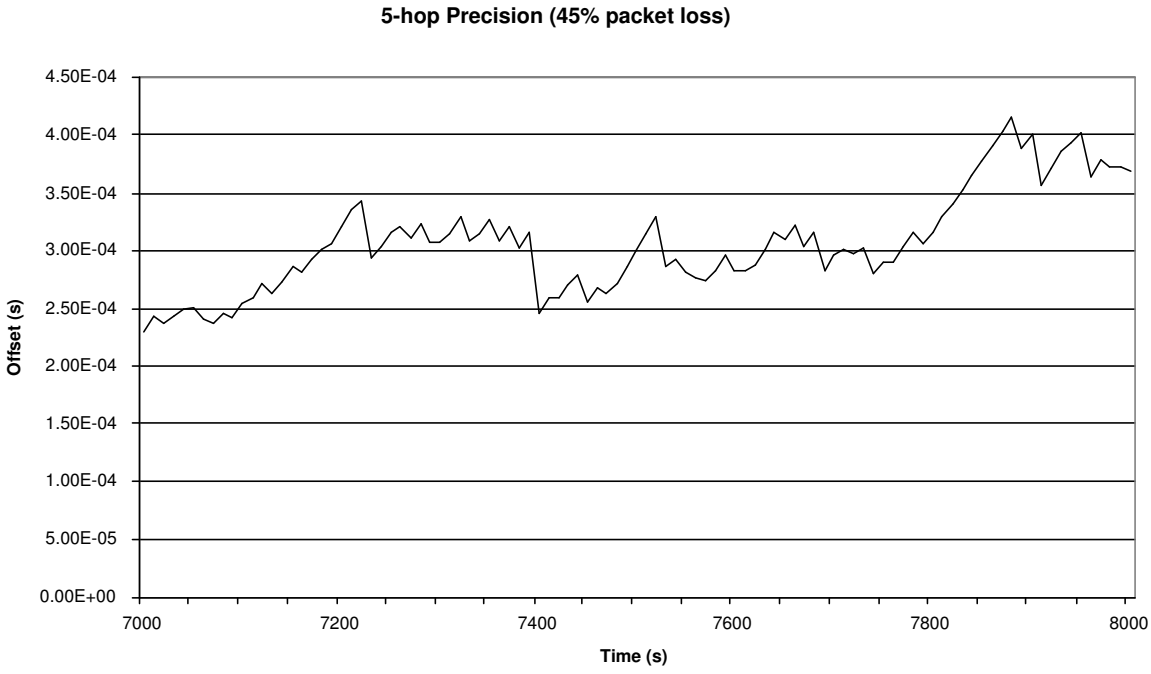


Figure 5.13: 5-hop precision

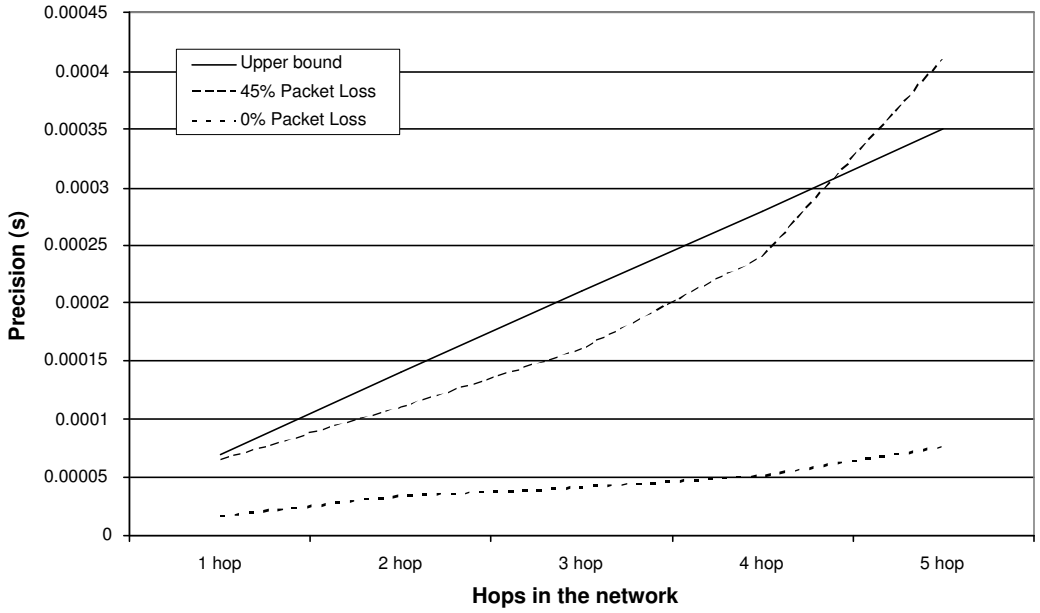


Figure 5.14: Synchronization precision in differing topologies

the randomness of the simulator. Suffice it to say that the accuracy of the clocks is not conserved in either environment, despite the fact that their average value is the Physical time upon initialisation.

Perhaps of more interest are the graphs shown in Figure 5.16. These show the accuracy of the system once it has reached its steady state in terms of precision. Both graphs approximately the same drift rate from Physical time. It is likely, however, that the synchronized clocks will drift from real time at the average drift rate of the local clocks. A study of this is beyond the scope of this project.

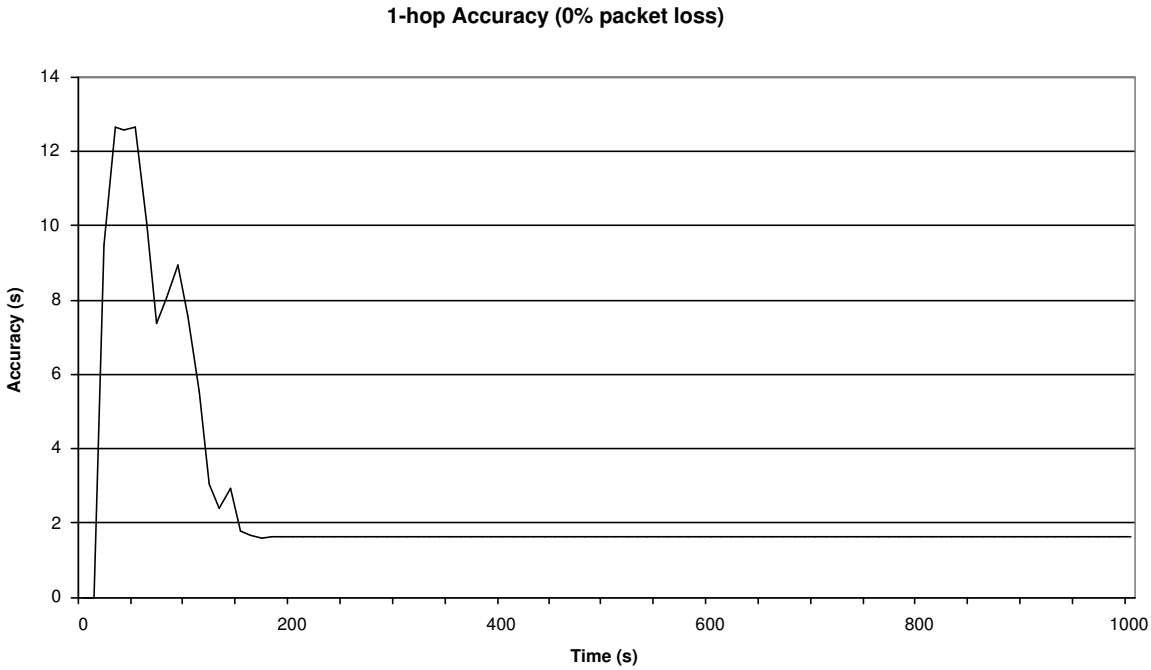
5.7 Summary

This chapter evaluates the tests which were carried out on this simulator. A number of fixed topologies were simulated in order to examine the effect of increasing network diameter (number of hops) on the precision achieved. The effect of a network partition on the algorithm was also simulated. Each test was carried out in both packet loss and non-packet loss environments. This allowed a comparison of the protocol under different conditions.

The effect of increasing message loss and network diameter effects the speed at which synchronization occurs. Steady state is reached where the only oscillation in precision is due to processing variance and time between resynchronizations.

Once steady state has been reached, the protocol achieves a high degree of precision where messages are not lost. This is consistently below the mathematical bound placed on it. Increasing the diameter of the network decreases the precision but its value is still less than the predicted upper bound for that topology. Message loss has a serious effect on precision due to increased time between resynchronizations on individual nodes. This results in the mathematical bound being violated as message loss increases.

A token evaluation of the accuracy of the algorithm was carried out for the sake of thoroughness. This revealed that during the early stages of synchronization prior to the network achieving a steady state, the accuracy was in no way preserved by the algorithm. Once a steady state had been reached, the nodes were observed to drift at a consistent rate from Physical time.



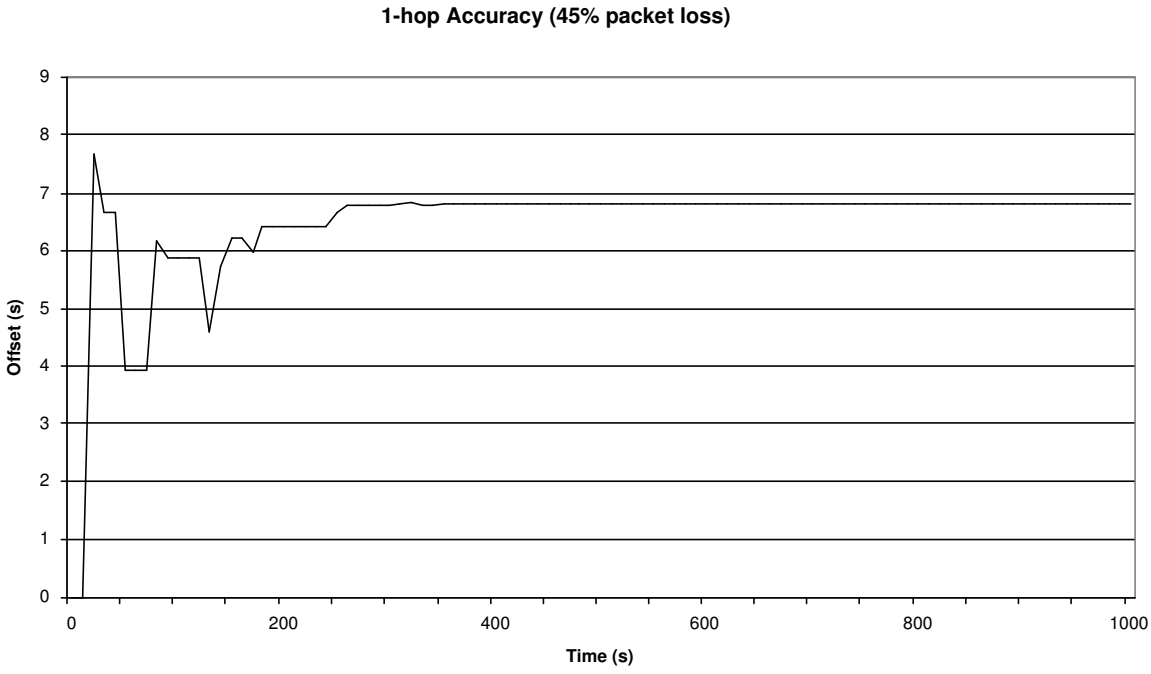
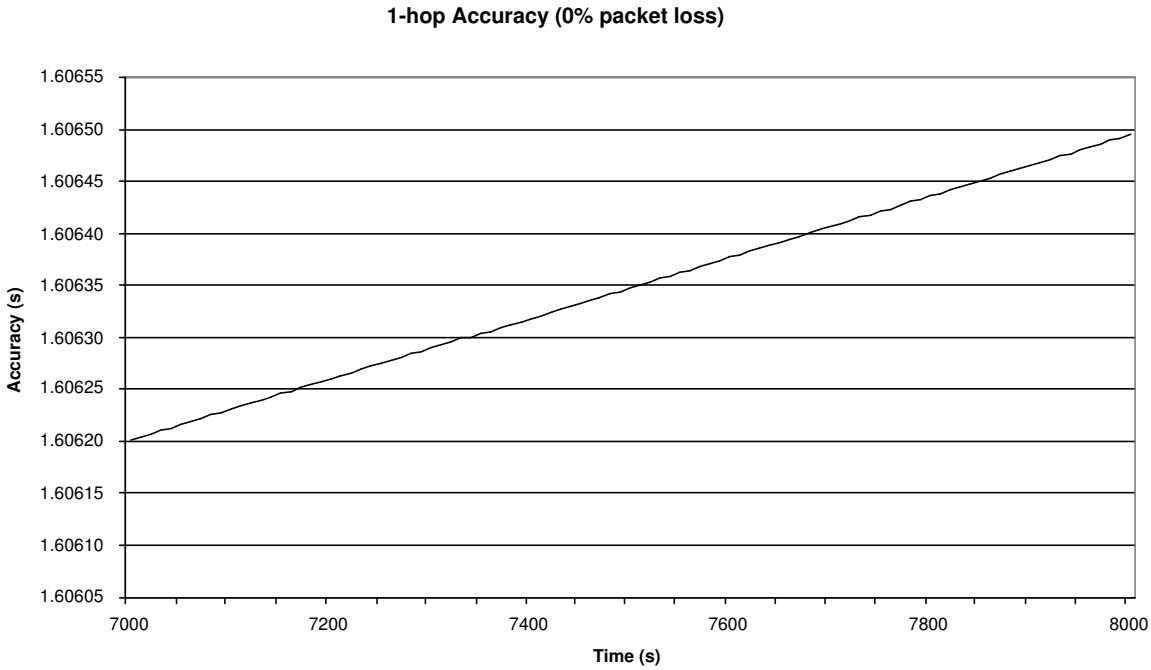


Figure 5.15: 1-hop accuracy



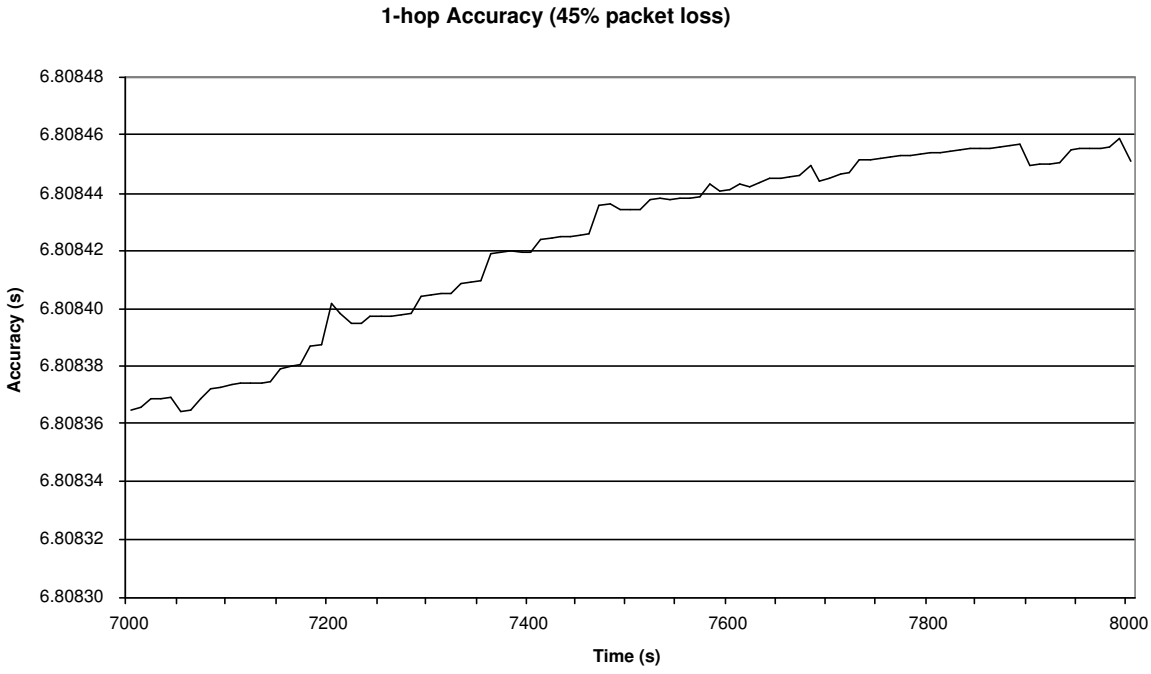


Figure 5.16: 1-hop Accuracy Steady State

Chapter 6

Conclusion

Research into the area of protocols for multi-hop wireless networks revealed that existing protocols did not achieve their full potential in terms of precision. This protocol addresses this problem by utilising the tightness property of the wireless medium to completely remove the propagation delay from the time-critical path. The result is a time-critical path which consists only of the difference between the maximum and minimum processing time taken to receive a message and create a timestamp.

In addition to providing more precise synchronization than other protocols for multi-hop wireless networks, this algorithm also reduces the prerequisites of other algorithm. Other algorithms which address multi-hop synchronization rely on the availability of multiple transmission channels and predefined transmission slots respectively. By comparison, this protocol relys on the presence of at least three nodes in a subset, which is required in order that the propagation delay be removed from the time-critical path.

The requirement that consecutive messages be received in order that synchronization take place has an effect on the fault tolerance of the protocol. Other protocols in the area achieve one-shot synchronization which means that a single message receipt provides a local clock reading. This is very fault tolerant. The fault tolerance issue of the presented algorithm is discussed further in Section 6.2.

6.1 Completed Work

- A new clock synchronization protocol has been presented for multi-hop ad hoc networks.

- A mathematical bound has been placed on the synchronization precision achievable between one-hop nodes which synchronize using this algorithm.
- The mathematical theory behind averaging, which extends this single hop synchronization to handle multi-hop environments, has been formalised.
- The operation of the protocol has been modelled in a simulator and evaluated.

6.2 Future Work

Time constraints ruled out the possiblilty of implementing this algorithm for practical use. It would also have been difficult to find resources which would permit the large scale testing which was carried out in the simulator. However it would be extremely beneficial to observe how well the simulator models the operation of the algorithm in a real world implementation. In addition, due to values for processing time being uncertain, the synchronization precision achievable is unknown. An implentation would provide the actual value for this when tested.

The issue of fault tolerance is of major significance in wireless communication. As nodes move towards the limit of their communication range, message loss increases dramatically. The probability of a pair of messages being received in subsequent rounds, as required by the algorithm, is $(prob(dis))^2$ where $prob(dis)$ is the probability of a message being delivered to a receiver which is dis distance away. This problem may be possible to address using an idea used from the algorithm described in [11]. This algorithm includes timestamps for n previous rounds in each synchronization message transmission, as opposed to timestamps for just the previous round as used in the proposed algorithm. These timestamps would have to be adjusted after each resynchronization as discussed in Section 3.2.1. This would allow resynchronization if messages were received less than n rounds apart.

The tightness property of is an assumption. The theory of relativity dictates that because the speed of light is fixed, there has to be a time difference between the arrival of a message at different distances from the transmission [5]. Therefore receivers which are not precisely equidistant from the transmission will not receive the message at the same instant. It may be possible to evaluate how far apart nodes are based on the power of transmission versus the power of receipt of a message. Possibly receivers could compare the power of the messages received as well as their timestamps for the messages, to evaluate exactly how tight the receipt was.

Finally there is the issue of round length. The length of time between resynchonizations has two effects. A longer round length means that it takes a longer Physical time to achieve the synchronized state. The round length also has an effect on the precision of synchronization as the local clock drifts between resynchonizations. However a shorter round increases the likelyhood that collisions will occur between synchronization messages, and also increases the amount of bandwidth needed to support the algorithm (one message is transmitter per node per round). It would be beneficial to evaluate this algorithm with varying round lengths in order that this balance be optimized.

Bibliography

[1] M. Lott A. Ebner, H. Rohling and R. Halfmann. Decentralized slot synchronization in highly dynamic ad hoc networks. Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC'02), Honolulu, Hawaii.

[2] F. Cristian and C. Fetzer. Probabilistic internal clock synchronization. In *Symposium on Reliable Distributed Systems*, pages 22–31, 1994.

[3] F. Cristian and F. Schmuck. Continuous clock amortization need not affect the precision of a clock synchronization algorithm. pages 131–143, 1990.

[4] D. Millinger A. Schedl H. Kopetz, A. Krüger. A synchronization strategy for a time-triggered multicluster real-time system. In *Symposium on Reliable Distributed Systems (SRDS '95)*, pages 154–161, Los Alamitos, Ca., USA, September 1995. IEEE Computer Society Press.

[5] S. Hawkings. A brief history of time. Bantam Books, 1988.

[6] IEEE. Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE Standard 802.11, 1997.

[7] H. Kopetz. Real-time systems: Design principles for distributed embedded applications, 1997. Kluwer Academic Publishers, 1997.

[8] L. Lamport. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM*, 21(7):558–564, 1978.

[9] A.M. Law and W.D. Kelton. Simulation modeling and analysis, 3rd. edition. McGraw-Hill, 2000.

[10] B. Liskov. Practical uses of synchronized clocks in distributed systems. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[11] E. Nett M. Mock, R. Frings and S. Trikaliotis. Clock synchronization for wireless real-time applications. 2000.

[12] E. Nett M. Mock, R. Frings and S. Trikaliotis. Continuous clock synchronization in wireless real-time applications. In *Symposium on Reliability in Distributed Software*, pages 125–132, 2000.

[13] D. L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*. 1994.

[14] E. Sourour and M. Nakagawa. Mutual decentralized synchronization for intervehicle communications. pages 272–277, 1996.

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

[16] P. Veríssimo and L. Rodrigues. A posteriori agreement for fault-tolerant clock synchronization on broadcast networks. In Dhiraj K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, pages 85–85, Boston, MA, July 1992. IEEE Computer Society Press.