Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

An Evaluation Framework for Time Synchronization in Lossy Links

Master's Thesis

Konstantinos Chaloulos

kchaloul@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:

Dr. Thanikesavan Sivanthi (ABB Corporate Research Switzerland)
Dr. Dacfey Dzung (ABB Corporate Research Switzerland)
Prof. Dr. Roger Wattenhofer

April 14, 2013

Acknowledgements

I would like to mention some individuals, who made the completion of this thesis possible and to whom I will always be grateful for their contributions.

I am grateful to all my supervisors at ABB for their guidance. My utmost gratitude goes to Dr. Thanikesavan Sivanthi for his most valuable guidance and support throughout this master thesis. From the beginning until the very end, he has always been there to discuss any issues occurred. This thesis would not have been possible without the contribution of Dr. Dacfey Dzung and Dr. Yvonne-Anne Pignolet, who followed with great interest the progress of my thesis and always provided interesting feedback.

I would also like to thank very much my tutor Prof. Dr. Roger Wattenhofer for offering me the possibility to pursue this thesis in ABB and for his supervision, providing valuable feedback and confirming the decisions made.

For their unconditional support throughout the whole master I would always be grateful to my family; my parents, Sarantos and Evanthia, who have always encouraged me, in many ways, to pursue my dreams and my brother, Giorgos, for supporting me, inspiring me, consulting me, advising me and being there for me from the first moment I came to Switzerland.

Abstract

Although wireless sensor networks have been widely studied, the special characteristics of these networks provide challenges from many points of view. One such challenge is time synchronization. The design and the performance of time synchronization protocols in wireless sensor networks have been examined analytically and using simulations for different network scenarios. However an empirical analysis of their performance under lossy conditions has not been carried out yet. In this work, a framework which consists of all components required to evaluate the performance of time synchronization protocols for multi-hop wireless sensor networks under lossy conditions is presented. The framework, which is built using Redwire Econotags and the Contiki operating system, is used for a show case evaluation of the two well-known synchronization protocols GTSP and FTSP. Both random packet losses and node failures are considered in order to provide some insights into the influence of lossy environments on the performance of time synchronization protocols for multi-hop communication.

Keywords: Evaluation Framework, Time Synchronization, Lossy Links, Wireless Sensor Networks, Multi-hop Communication.

Contents

Chapter 1

Introduction

Several applications require a time synchronized network, in order to be able to order events, to measure the time interval between events or to simply provide information about which time of the day an event happened, on specific or different nodes of the network. The time synchronization protocols try to achieve a common notion of time among the nodes in a network.

The focus of this thesis is on wireless sensor networks (WSNs). Several special characteristics differentiate these networks from common computer networks. WSNs are composed of nodes with limited computing and communication capabilities. WSNs are used to sense some physical phenomena and to communicate the relevant information to a base station. Typical applications of WSNs are in areas, which are not physically accessible and where wired communication is impracticable e.g. for forest fire detection.

Although time synchronization in WSNs is a significant requirement for many applications, the motivation behind this thesis is to be able to deal with the fault detection in the power distribution grid. In Figure 1.1 it is shown, how time synchronization can achieve the protection of the power network by a faulty line.

At the two ends of the line two sensors, powered by the grid line, can be set, which take samples of the current running through the line. The two samplers can assign a timestamp for each sample and based on those timestamps detect a fault and isolate the affected lines by turning some switches, i.e. circuit breakers, off. In order for those sensors to be able to detect a fault, e.g. an overcurrent, they have to be synchronized so that the samples collected can be assigned to the same phase of the waveform of the current. Hence the two sensors have to communicate in order to maintain a common notion of time. While extending this example to many more measuring points and grid lines, the sensors have to synchronize to each other, communication over multi-hop topologies. Moreover the links suffer often from losses, due to the unreliable nature of the low-power wireless links.

For achieving this time synchronization in WSN, several time synchronization protocols are proposed. Typical approaches used for other type of networks are

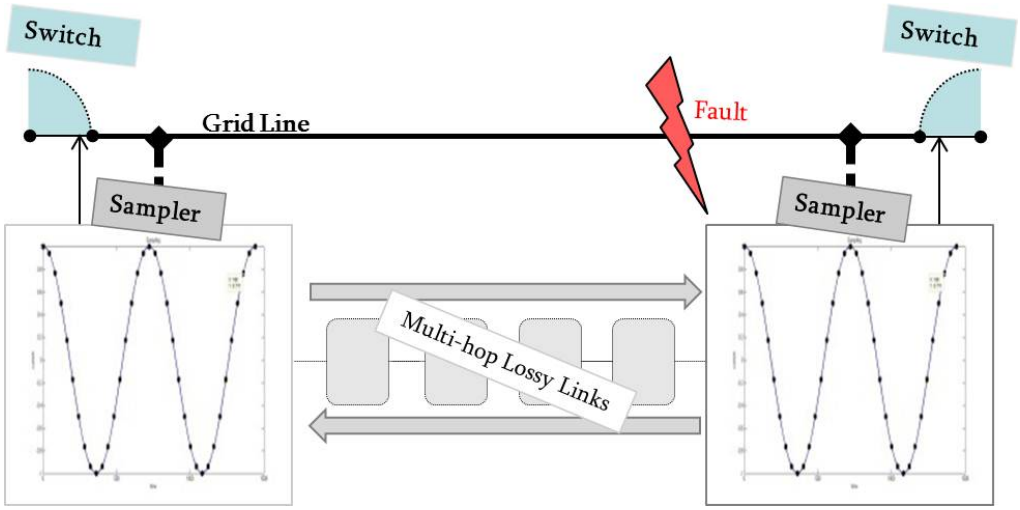

Figure 1.1: Grid line fault detection utilizing time synchronized samplers exchanging information over a multi-hop lossy network.

often not applicable, because such networks have some special features that have to be addressed. The hardware used for such nodes is usually restricted to basic elements, with the main objective to reduce cost because the number of nodes to be deployed is very large. Furthermore, the bandwidth and energy limitations restrict the protocol designers, whose goal is hence to achieve the best possible synchronization, utilizing the limited resources available in an optimum way.

The choice to reduce the cost of the platforms used comes with a further tradeoff. The crystal oscillators, which are used to trigger clock ticks, tend to be quite unstable, resulting in clocks whose time progresses at different rates. Syntonization, where the electronic circuits are adjusted in order to operate at the same frequency, is not a practical solution. Even if nodes are syntonized before they are deploys, external variant parameters can still result in the nodes' deviation from the nominal frequency. The most dominating factor is temperature. Hence any time synchronization protocol has to provide a software based solution, i.e. drift compensation algorithm, which can assess the differences in the clock rates and compensate for the resulting clock offsets. The introduction of software-based elements adds variance and delays, since the time-related information contained in the message exchanges needs to get through all the layers of the network stack of the operating system of the node.

One more special challenge is the fact that the network connections are unstable[1]. The limited communication range of the sensors, the possible external interference due to the unprotected nature of the wireless medium leads to high bit error rates and message loss. The hardware used is likely to be subject to radical ambient changes depending on the environmental conditions. This leads to unpredicted behavior of the hardware. Moreover the nodes can

be obstructed by natural objects. Such cases are translated, in terms of communication, into bursts of packet losses. The losses deteriorate the performance of time synchronization protocols, increasing the synchronization errors between nodes.

The goal of this thesis is to define a framework for evaluating the performance of time synchronization protocols in multihop networks with lossy links. In particular to analyze the different factors (e.g. OS delays, oscillator stability, packet losses etc.) which affect accurate synchronization, to propose methods to mitigate these effects and to implement and evaluate the proposed methods in a test bed.

The rest of the thesis is organized as follows: Chapter 2 presents the related work, providing a short overview of time synchronization protocols and related efforts made to evaluate their performance from different point of views. Chapter 3 gives a brief introduction to the target platform, the operating system and the existing constraints, which limit the freedom of choices for the test-bed definition. In Chapter 4 the framework is presented. The methodology to measure the synchronization offsets and error of the network, the extensions required to introduce losses, the mitigation techniques to improve the performance of the protocols, the configuration parameters and the evaluation metrics are discussed. Chapter 5 presents the framework at work, evaluating two synchronization protocols, FTSP and GTSP. Chapter 6 includes the conclusions and proposal for future work. In Bibliography, a list of related references is provided.

Chapter 2

Related work

This chapter gives an insight into some state-of-art time synchronization protocols and efforts made to evaluate their performance.

2.1 Time Synchronization Protocols

Time synchronization has always been considered to be a complex and challenging issue, when dealing with wireless sensor networks. Although the solution using GPS pulses to synchronize achieving tight synchronization in the order of picoseconds does solve the problem in theory, practical issues exist making this choice undesirable. The cost of having hundreds of sensors equipped with GPS receivers is high and the energy consumption required would result into limiting significantly the duration of life for the sensors. Furthermore the GPS signals are not available inside buildings or in general covered areas, where the line of sight to the GPS satellites is not available.

Moreover the well-known synchronization algorithm Network Time Protocol (NTP) [2] is not practical for time synchronization in wireless sensor networks, since each node should run an NTP client and connect to an NTP server in the internet in order to synchronize, which is an overkill for WSN nodes. Moreover the achievable accuracy is limited in the order of microseconds, accuracy which often does not meet the requirements of the target applications. Due to the special characteristics of the wireless sensor networks, namely the limited energy, computational power and bandwidth available, several solutions in direction of achieving network synchronization are proposed, some of which are briefly presented below.

The Reference Broadcast Synchronization (RBS) [3] protocol takes advantage of the fact that the variance of the propagation delay in the wireless medium is negligible and achieves synchronization with a two-step procedure. Initially a node broadcasts a message. The nodes receiving the message timestamp it and as a next step they exchange the timestamps between them. Since the propagation delays in WSNs are in the order of nanoseconds, the time difference between the

reception of the message by the two nodes is negligible. This provides the nodes with a reference point of time, whose timestamp is used so that the receiving nodes can come to an agreement about a common notion of time.

Different approach is used by the Timing-sync Protocol for Sensor Networks (TPSN) [4], which builds a spanning tree with a root node, whose time is used as reference in order to synchronize the whole network. The protocol uses the two-way delay measurement technique, in order for each node to be able to calculate the delay and the relevant offset compared to its parent. Based on this information it corrects its local time so that it synchronizes with its parent.

The Flooding-Time Synchronization Protocol (FTSP) [5] utilizes a similar technique, where once again the time of the root is distributed to the network. The authors deal for the first time with the delay and its variance occurring in the message exchanges, proposing the solution of timestamping at the MAC layer. Moreover the nodes are keeping track of their relative offsets from the root in order to calculate their relative drifts and compensate for them. The protocol will be studied in detail in Section 4.1.2.

The Routing Integrated time synchronization protocol (RITS) is also a tree based time synchronization protocol [6]. The difference is that the synchronization is initiated upon the detection of an event. The event is timestamped by the detecting node and further forwarded towards the root nodes. Each node converts the timestamp from the senders local time to its own, until the timestamp reaches the root.

Another tree based synchronization protocol is Glossy [7], which exploits the constructive interference of parallel transmissions to achieve a quick network flooding. The timing requirement in order to achieve this constructive interference is examined and it is shown how implicitly the nodes can achieve time synchronization, based on calculation of their distance in hops from the tree root and the delay per hop.

Totally distributed approach is used by Reachback Firefly Algorithm (RFA) [8], where the way that neurons and fireflies synchronize is applied to synchronize the network nodes. Each node uses an algorithm to align its own firing phase, based on the firing phases of other nodes. However the synchronization is only focused on synchronize some periodically generated pulses, without achieving a common notion of time for the nodes of the network, providing synchronicity and not time synchronization.

Also distributed is the Gradient Time Synchronization Protocol (GTSP) [9]. The synchronization algorithm used by GTSP utilizes two time-related information, namely the absolute value of the time and the rate, at which the time progresses. Each node, when receiving a message from a neighbor, saves their rate and the offset between it and the neighbor. When a synchronization period is expired, a node uses the saved information. It averages the rate and the off-

sets with its neighbors, corrects its rate and time and broadcasts its time-related information. The protocol will be studied in detail in Section 4.1.2.

2.2 Evaluation Efforts

Apart from the efforts of creating time synchronization protocols for WSN, there are several publications considering effort towards their evaluation.

In [10] a comparative survey between different time synchronization protocols is presented, based on several factors. The authors studied several protocols, highlighting design goals, advantages and disadvantages, which they present for each protocol both its qualitative and its quantitative performance. Among the factors examined for the quantitative comparison are the protocol's precision, complexity, convergence time, network size, and even the existence of GUI services. For the qualitative comparison accuracy, energy efficiency, complexity, scalability and fault tolerance are among the metrics evaluated.

A more theoretical approach on the clock synchronization and the problems that remain still unsolved was presented in [11]. The authors point out that many evaluations are based on the worst case scenarios, creating models that are often too pessimistic compared to actual situations, resulting in an improper representation of the real applications. Furthermore they examine dynamic networks, showing that the asymptotic bounds for worst-case clock skew calculated might not be achievable in dynamic networks, where neighbors may leave the system at critical moments. The introduction of a model for faulty systems, either with lossy and unreliable links or with faulty nodes with a certain probability, combined with density of the graphs and the fault-tolerance of a protocol is assessed to be still open for further research.

A review of clock synchronization protocols used in wireless sensor networks is presented in [12]. The authors investigate methods for estimating the clock offset and skew and present an analysis of two synchronization protocols; TPSN and RBS.

Although several directions of evaluating several aspects of time synchronization in wireless sensor network exist, the effect of lossy links on the performance of time synchronization protocols is not analyzed. Most of the approaches are either theoretical or include also some simulation results. The main contribution of this work is a framework, which analyzes and implements the components required in order to evaluate the performance of time synchronization protocols in multi-hop WSNs on hardware. The components of the framework support the evaluation in a network which suffers from packet losses and node failures. The use of the framework is then demonstrated in practice, with the implementation and evaluation of two well-known protocols, FTSP and GTSP, for different lossy conditions.

Chapter 3

Platform

The platform to evaluate the performance of synchronization protocols is the Redwire Econotag [13]. The embedded operating system for the target platform is chosen as Contiki OS. The Contiki OS is a cross-platform operating system and comes with the Instant Contiki development environment, which has all necessary tools required in order to start programming the Econotags.

3.1 Redwire Econotag

The Econotag (Figure 3.1) is based on the MC1322x family of Freescale's third-generation ZigBee platform. The main features of this product include the 802.15.4 compliant on-chip transceiver or modem which supports signaling with 250kbps data rate, a 32-bit ARM7 CPU core with programmable performance of 24MHz and 128Kbyte serial flash memory, 96Kbyte SRAM and 80Kbyte ROM.

Figure 3.1: Redwire Econotag.

3.2 Contiki OS

The "Contiki OS is an open source operating system for the Internet of Things"[14]. A wide variety of applications and systems can use this operating system, since it is designed to run on a range of platforms. Among the main features of Contiki is the support of IPv6 and IPv4 along with several recent low-power wireless standards, such as 6lowpan, RPL and CoAP. The language to develop an application is standard C, while the typical size of a binary to be loaded is in the size of few tens of Kbytes.

Contiki is designed to be a cross-platform operating system. For this purpose there is a separation of its code to some generic modules and some platform dependent functions. The generic modules implement the functionalities that any platform should support, such us the network stack and the communication at the network layer, while the platform dependent functions deal with hardware specific tasks, such us the transmission and reception of a packet.

Contiki is a hybrid model, combining an event-driven system and pre-emptible threads. It uses an event-driven kernel, while the preemptive multi-threading is an application library. The usage of this library is optionally linked with programs that explicitly require it [15].

3.3 Constraints

Before the design and creation of the framework, several constraints had to be identified. Those constraints arise mainly due to practical and technical limitations, that led to a specific test-bed definition. The limitations are discussed below.

3.3.1 Internal Synchronization

Clock synchronization is performed, in order for the nodes of a distributed system to correct their local time information. There are two main ways to achieve this synchronization; the internal synchronization, where the nodes synchronize to a common time, and the external synchronization, where nodes achieve not only internal synchronization, but synchronize also to an accurate external real-time clock source, such as the Universal Coordinated time (UTC).

In order to achieve external synchronization, at least one of the nodes of the network has to be provided with the real-time clock source. Redwire Econotags have two communication interfaces, the wireless and the USB interface. Since the wireless interface is used for the time synchronization protocol message exchanges, the second interface could be used for providing the real-time clock.

In [16] the latency of the USB interface is studied for different real-time applications. They show that the uncertainty of the delay in the communication over the USB interface ranges in the order of milliseconds, which is orders of magnitude worse than the accuracy achieved by the synchronization protocols. Hence, the evaluation framework considers only internal synchronization, which is enough for ordering events and evaluating the performance of the protocols.

3.3.2 One Broadcast Domain

The choice of the platform implies some limitations for the test bed. More specifically, the Redwire Econotags used do not provide build-in support for external battery supply. [17] explains a step to step procedure needed for connecting an external power supply to Econotogs. However, since for the line fault detection application the Econotag can be powered from the power grid network, it was decided to avoid any modifications on the platform and use the power over USB for the test bed setup.

Using the USB interface to power up the nodes creates some limitations on the physical placement of the nodes. The commercial USB extension cables reach up to approximately 5 meters. Hence the maximum achievable distance between any two nodes connected to the same computer via USB is around 10 meters. On the other hand the communication capabilities of the nodes extend much further, achieving successful communication for nodes which are up to 30 meters apart. As a result the nodes will belong to one collision domain, creating a complete mesh network.

In order to be able to evaluate different network topologies, each node maintains a whitelist of node IDs. Only packets from senders on the node's whitelist are considered for time synchronization. By means of this whitelist any kind of network topology can be simulated, but without the characteristics of lossy environments.

3.3.3 Time Resolution

The Econotags operate at 24MHz clock frequency and a special module implementing a timer is provided. Unfortunately, since the sensors are not primarily targeted to keep tight synchronization in the orders of microseconds, the resolution of this timer is per default 10ms. The design of the timer module provides some parameters which can be modified, in order to achieve a better resolution. The implementation of the timer is based on an interrupt produced every time when an increasing counter reaches a comparison value. Hence, in order to improve the precision, one has two options. Either to decrease the value to which the counter is compared or to change the frequency by which the counter increases its value. With such parameter configurations the best resolution achiev-

able of this counter is 40µs. Going below this limit makes the system unstable, due to the high frequency of interrupts produced.

Since 40µs is not good enough to evaluate protocols whose synchronization error is in the range of few microseconds, an alternative timer is used. The Econotag has the MACA module which implements the MAC layer for the platform. In this module a timer, i.e. the MACA timer, is used for the transmission and reception of packets. This timer is compliant with the 802.15.4 standard, achieving transmission rate of 250kbps, leading to a 4µs resolution. In the next section it is shown how this precision limits the performance of the protocols, introducing errors every time that a timestamp has to be created.

3.3.4 Performance Limitation

The resolution of the timer, i.e. 4µs, imposes some limitation on the best possible achievable synchronization. The execution of the commands is done in time steps much smaller than the 4µs of accuracy measurable. Hence the timestamps upon a single transmission and reception suffer from errors, depending on which relative point of time during a 4µs cycle of the MACA clock is the command issued.

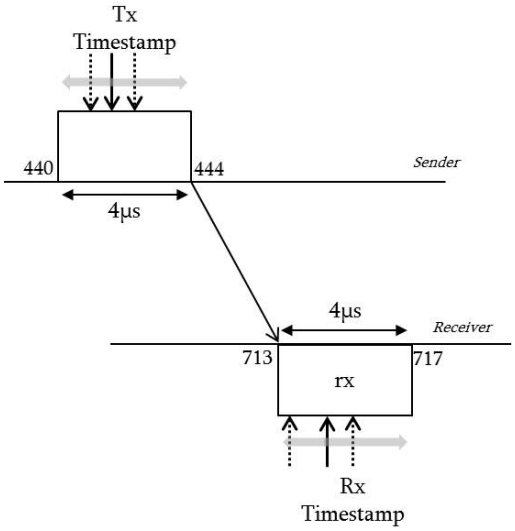

Figure 3.2: Example of error inserted by the limited timer resolution: The timestamp command can be issued at any time during the interval of 4µs, leading to errors which limit the performance of time synchronization protocols.

In Figure 3.2 a simplified example of the timestamping during transmission and reception of a packet is presented. The timestamping can be issued at any point of time during the 4µs of a clock tick, both for the sender and the receiver. In all cases, the timestamps will be read from the counter's values,

which in this example is 440 for the sender and 713 for the receiver. Hence, the error introduced by the timestamping is assumed to be equally distributed in the range [0,4)µs.

Both time synchronization protocols evaluated provide drift compensation mechanisms, responsible for dealing with the slightly different rates at which nodes increase their time. Although those mechanisms can deal with the mean of this error, its uncertainty cannot be addressed, thus influencing the overall synchronization performance. For our platform, the variance can be calculated: the platform runs at 24MHz frequency, which is translated into steps of 42ps, or 0.042µs. The standard deviation of an equally distributed discrete variable in the range [0,4)µs with step 0.042µs is σ_0=1.17 µs.

The error occurring during timestamping of the transmission and reception of one message, as well as the one occurring at each separate message exchange at the next hops are assumed to be uncorrelated. Based on the propagation of uncertainty, for uncorrelated variables, whose covariance equals to 0 and standard deviation σ_0, the uncertainty in the final variable, expressed in terms of standard deviation, is given by 3.1. Assuming the standard deviation similar for all devices (based on the uniform distribution of error), then every transmission or reception of timestamps is increasing the uncertainty. As an example, for a 9 hop network, where the time reference is the root, 8 transmissions and 8 receptions are needed to be performed until the time reaches the furthest nodes. In total the variance of these 16 uncorrelated events is added up, leading to an error of standard deviation σ_8:

$$\sigma_8 = \sqrt{n} \cdot \sigma_0 = \sqrt{16} \cdot 1.17 = 4.68\mu s \tag{3.1}$$

However in practice, it is difficult to estimate what error would be expected for the two protocols used. Furthermore, apart from the error resulting by the limited time resolutions, more factors, such as the transmission and reception delays and their uncertainty, increase the synchronization error between nodes separated by several hops.

In order to address the influence of the existing uncertainty in the multi-hop network to be evaluated, experiments are performed to measure the standard deviation of the synchronization error for 8 hops. The reference time is broadcasted every 1 second by the root and each node is informed by its parent. In Table 3.1 the mean value and standard deviation of the synchronization offsets between nodes which are placed 1 until 8 hops away from the root are gathered:

Although the mean of the offsets can be calculated and compensated for by the drift compensation algorithms, the uncertainty of those values as expected increases after every hop. Moreover the uncertainty of the offset of the node 8 hops away from the root, expressed in terms of standard deviation, is much higher than the expected uncertainty introduced by the error resulting from the

Hop(s) from root	1	2	3	4	5	6	7	8
Mean offset(µs)	-2.70	-3.77	-13.56	-8.19	-11.26	3.91	7.54	1.27
Std of offset(µs)	6.12	6.78	8.95	13.32	15.11	17.31	18.13	21.21

Table 3.1: Measured delays and standard deviation of the offsets between nodes n hops away from the root

low time resolution. In order to investigate the cause of this higher measured uncertainty, measurements concerning the hop delay and its variance are realized and discussed in the corresponding Section 4.3.3.

It has to be noticed that this error analysis is presented to show using a simple test example what further limitations can be imposed by limited resolution and how important they can be when evaluating a protocol.

3.3.5 Limited Number of Nodes

One more restricting factor is the number of nodes that can be supported by the system. Since the nodes are powered via the USB interface, they should be connected to the personal computer (PC). This imposes a technical limitation on the test bed. Although there are 3 USB ports at the PC used, their number can be extended by the usage of USB hubs. However, it is observed that the communication between the Econotags and the PC is more often broken, when the number of the nodes connected to a specific USB port of the computer is increasing. Hence only network topologies consisting of up to 10 nodes are considered in the test bed.

Chapter 4

Evaluation Framework

The framework consists of an integration of several components. The interoperability and co-function of all these components requires a careful design and implementation.

In Figure 4.1, in white background are those components which are necessary for the synchronization of the network, while in other colors are the extensions required, to measure the synchronization accuracy and introduce losses into the network. The whole framework can be separated into two major block components.

The first major block includes components related to the network synchronization operations. This component can further be subcategorized into the synchronization operations and the framework extensions. The synchronization operations include the broadcasting of time information and functions executed upon reception of such a message. The extensions on these functions support the event reporting, which is a method to get the local time of all nodes, and the losses, which is information used by the nodes to simulate lossy links.

The whitelist check, which is part of the network synchronization, is implemented right before the receive function of the protocols and affects the protocols by dropping their packets. The event reporting and the update of the variables concerning the lossy environment are framework extensions and are performed by an external node. This node is broadcasting different packets, which have different structure in the packet fields and are perceived by the nodes of the network as events or as updates of the loss conditions.

The second major block includes the processing of the data collected by the event reporting process. Initially some scripts, written in Linux bash scripting language, select the data of interest, excluding garbage and booting prints. Afterwards Matlab scripts read the provided data and extract information about the test performed. Moreover the scripts check for the validity of data by performing checks such as to find events not reported by all nodes. Finally after the checks are passed and any actions required are done, the plotting tools of Matlab are used, in order to evaluate the performance of the network synchronization

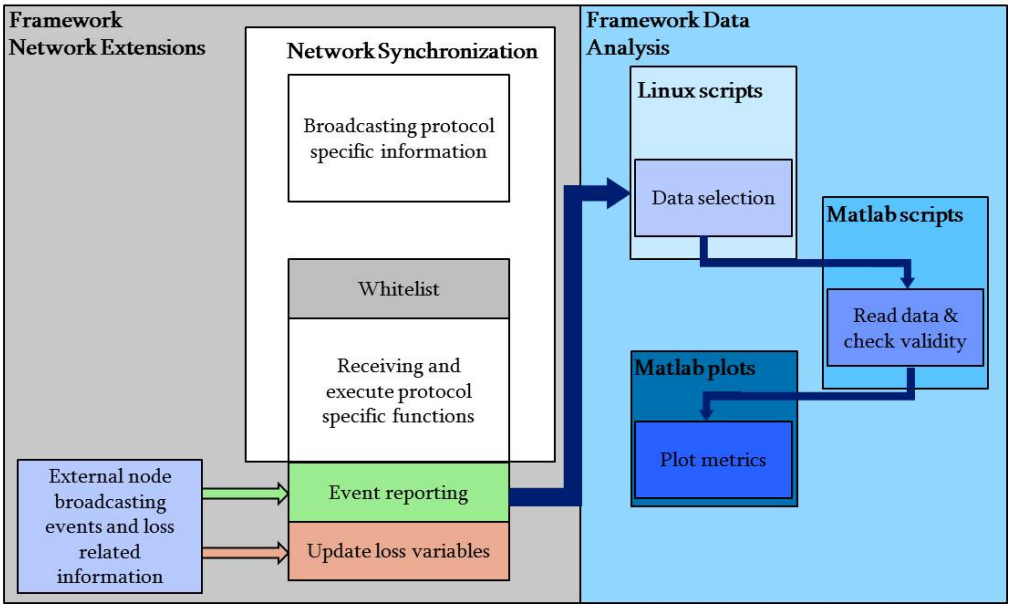

Figure 4.1: Evaluation framework: Its components and their integration.

for the scenario tested.

Each separate block is discussed further in the following subsections.

4.1 Setup

The full setup required in order to analyze the performance of a synchronization protocol contains a sequence of steps.

All nodes belong to a broadcast domain, meaning that all nodes can receive messages from all other nodes. Hence, virtual multi-hop topology can be created based on the symmetric whitelists.

In Figure 4.2 the blue continuous links show the allowed connections, while packets received from the dashed red links are dropped, creating a multi-hop environment. The blue links are considered for the synchronization protocol to be evaluated.

After exchanging some time sync messages, the nodes are assumed to be synchronized. As a next step an external node is introduced in the network, independent of any synchronization operation. The node broadcasts periodically newly introduced type of messages; either an “event” (Figure 4.3) or an “update” of the loss variable. The nodes, with the framework extensions, handle the “events” with the “events reporting” process, or update their loss variables, with an extension of the reception function. Since it is a broadcast with negligible

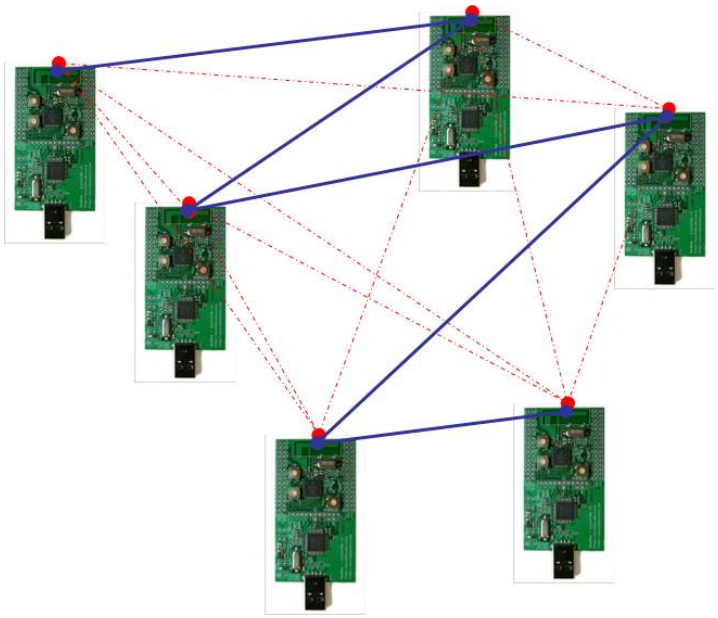

Figure 4.2: Multi-hop topology is created based on whitelists.

propagation time, all synchronized nodes should report the same event reception time. Any differences in the reported timestamps are thus measurements of the clock offsets.

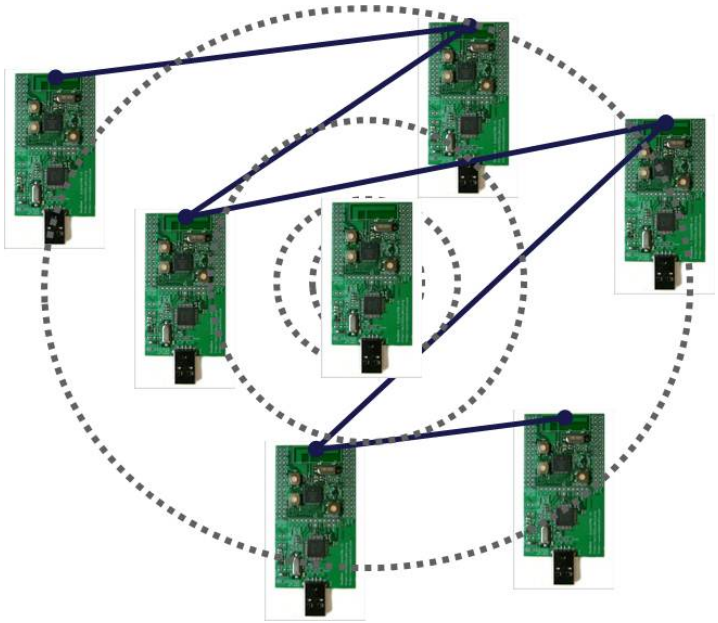

Figure 4.3: An external node is periodically broadcasting events.

In case of an “event” message, the nodes timestamp the packet’s reception and report the timestampscalling a separate process, through the USB interface to the PC connected, as shown in Figure 4.4.

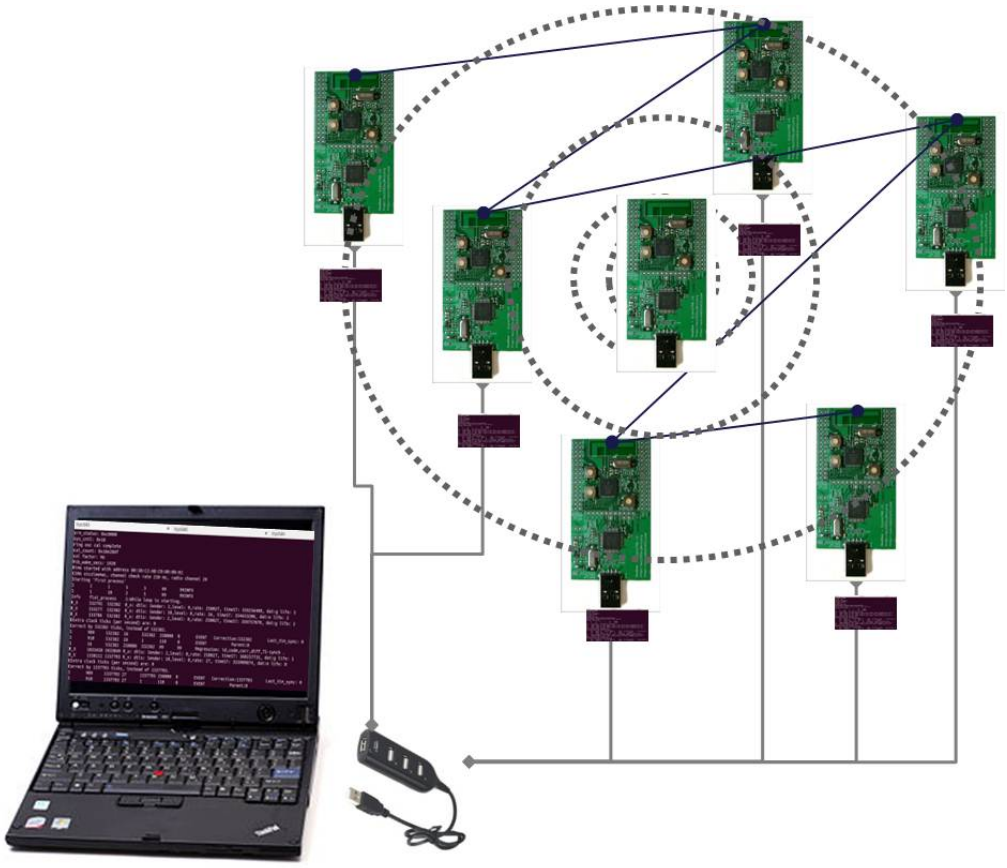

Figure 4.4: Nodes timestamp the events and report the timestamps to the PC.

The access to the file where the nodes print timestamp information has to be managed, in order to avoid overlapping prints. The access scheme to the file is chosen to be time division multiple access (TDMA). Each node is assigned a timeslot, during which it has to print the timestamp into the file.

After logging the timestamps, the necessary information to evaluate the performance of the synchronization protocol is available. Linux scripts are run on the file, to clean the data and keep only the necessary data in a tab-separated format. The file is passed to Matlab for further processing, as depicted in Figure 4.5.

Matlab initially reads some configuration specific information provided by the nodes. Then it performs some necessary checks, in order to make sure that the data provided are valid. After those checks, the calculations and plots are

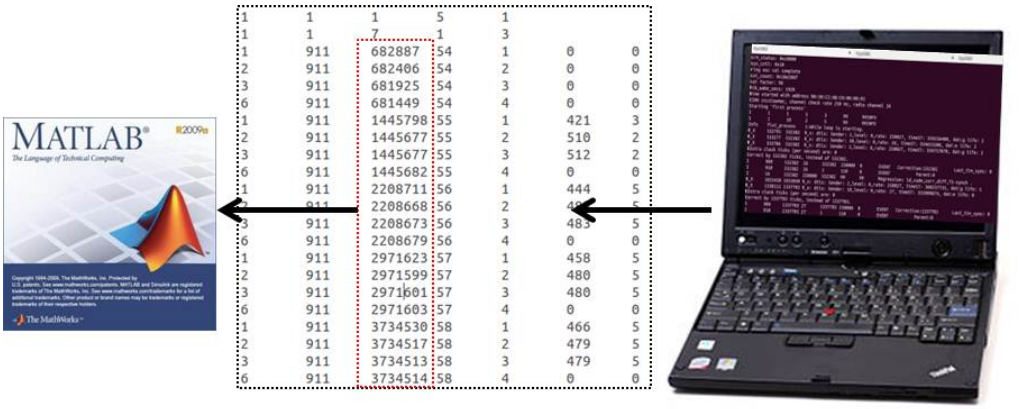

Figure 4.5: A tab-separated file contains the timestamps which are passed to Matlab for evaluation.

produced for the evaluation metrics of interest (Figure 4.6). Such metrics are the global and local synchronization offset.

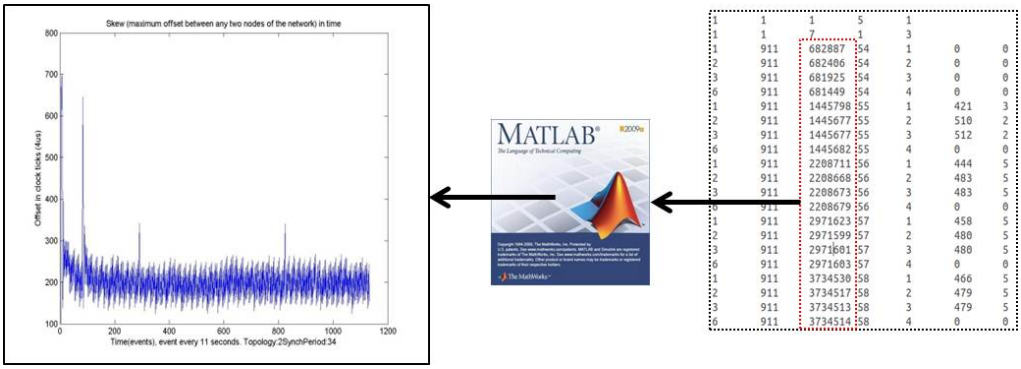

Figure 4.6: Matlab uses the timestamps to provide plots and statistics for evaluation.

The aforementioned steps are further examined in detail separately in the following sections, where the extensions required are presented and discussed.

4.1.1 Multi-hop Topology

One of the goals of the framework is to evaluate the influence of the diameter of the network, by testing different topologies. In order to be able to create a multi-hop environment, a method is implemented, which is creating links between nodes by setting up appropriate whitelists in each node. In this way, the user of the framework can choose which nodes can communicate with other nodes. In other words, the nodes are informed which are their neighbors. In order to

facilitate the user, some topologies, namely the mesh, the linear and the double linear topology, are predefined.

From a programming point of view, each nodes keeps in memory a whitelist of its the neighbors. Each message from a node that does not belong to the whitelist is ignored. In other words the information in the message is not used by any of the synchronization protocol since its main purpose is the creation of different network topologies.

Two topologies to be evaluated are implemented, with the help of the whitelists. More specifically:

- a linear topology, where nodes are placed on a line and each node can hear the previous and the next one, as depicted in Figure 4.7.

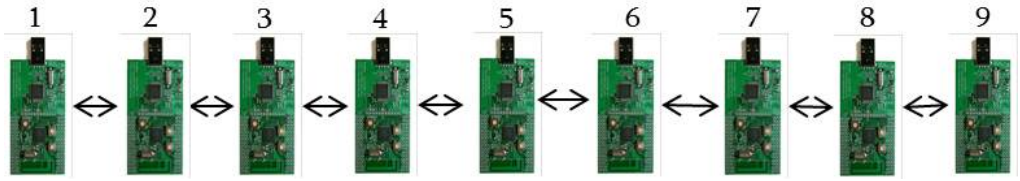

Figure 4.7: Linear topology.

- a double linear topology, where nodes are placed on two parallel lines and each node can hear at most two adjacent nodes on its line, plus at most two neighbor nodes of the other line, as depicted in Figure 4.8.

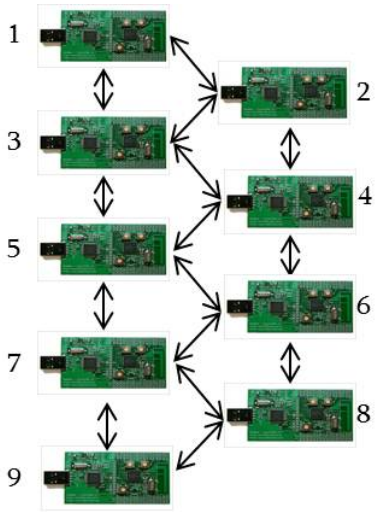

Figure 4.8: "Double line" topology.

The two topologies will be used for the evaluation of the synchronization protocols.

4.1.2 Synchronization Protocol

A time synchronization protocol is applied to the nodes. There are two separate components of the synchronization protocol. Firstly is the reaction of a node upon expiration of a timer, which signals the expiration of a synchronization period and secondly the reaction upon the reception of a synchronization message.

Every node, after booting is completed, sets a timer, which periodically expires. This timer is used for realization of time synchronization. The frequency with which the timer expires is equal to the synchronization frequency.

Contiki provides a set of timer libraries that can be used both by application programs and by the operating system itself. In the implementation, event timers (etimers) [18] are chosen. The event timers generate timed events. More specifically, when the event timer expires it will post an event to the process that set the timer. The function of the time synchronization protocols, which will broadcast time information is implemented as a process, which constructs and sets the etimer at the beginning and then blocks infinitely into a while loop. When the etimer expires, an event is posted to the process that implements the time synchronization protocol, realizing the core algorithm of the synchronization protocol and broadcasting time information, as depicted in Figure 4.9.

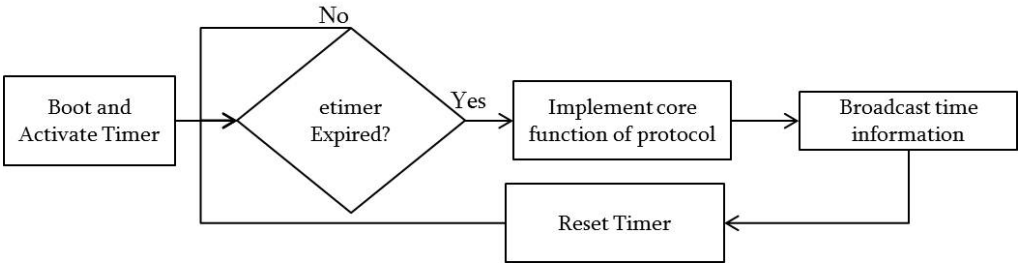

Figure 4.9: Event timer expires to realize the time synchronization protocol.

The second component of the protocol is the function implemented upon the reception of a time synchronization message. Using once more the event driven nature of Contiki, the reception of a message triggers an event handler, implemented as a function. Prerequisite for the reception of a message is the construction of a broadcast connection, which practically checks the frequency channel chosen for communication. The implementation of the protocol, which is done into this receive function as shown in Figure 4.10, typically includes storing time information which has been broadcasted by its neighbors.

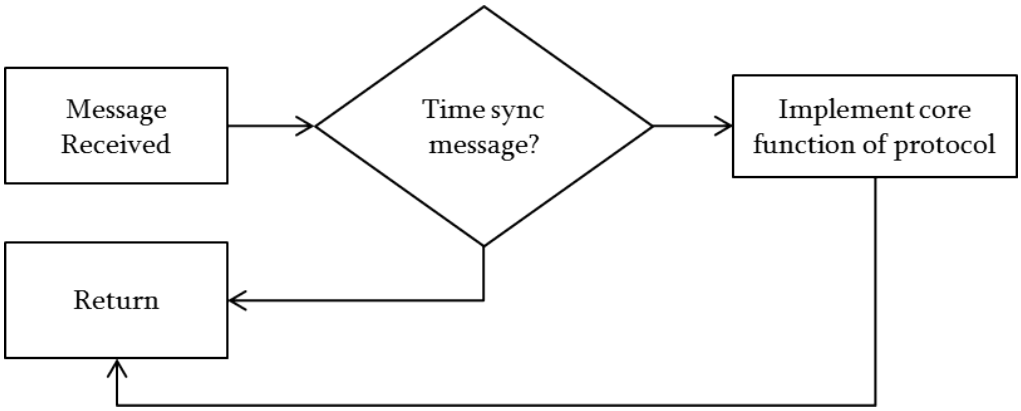

Figure 4.10: Functions of the protocols are implemented in the receive function.

Each protocol would differentiate from others based on how the time-related information is used in order to achieve a common notion of time with neighbors.

Flooding Time Synchronization Protocol (FTSP)

In this section, the core features of FTSP [4] are presented. The information included into the messages exchanged between nodes consists of three fields: the timestamp, the id of the node which is considered to be the root (rootID) and a sequence number (seqNum). The synchronization is achieved through a two-step procedure. Initially a root is elected. The root will be considered to have the correct time and all nodes will have to correct their time with respect to the root.

The protocol is designed carefully to deal with new nodes entering the network; any node is allowed to broadcast its time after it has collected some consistent reference points. In order for a node to collect those point, it must receive synchronization messages by the root, if it belongs to its broadcast radius, i.e. the neighbor list, or by any other synchronized node of the network, which has already collected those reference points. Hence a node that joins the network has to wait for an amount of time and get information about the existing time in the network.

The root election is done based on the exchange of the rootID field. Every node begins declaring itself as the root, unless it receives a message which includes a rootID field smaller than the node's id. The goal is that after some message exchanges, the node with the smallest id in the network will be chosen to be the root node. Resistance against racing conditions, where the root has disappeared but the nodes keep updating their root based on their neighbors information, is provided based on the sequence number field. This field is used to reject any outdated information received. Furthermore, a timeout is defined, so that a

possible disappearance of the root is detected and the election of a new root can start up again. The smooth transition from an old root to a new root is also taken into account, since each node is maintaining and using the drift information calculated, even after getting elected as the new root of the network.

Analyzing the reception of an FTSP sync packet into separate blocks is shown in Figure 4.11. Initially the node has to check its validity based on the sequence number, which is increased only by the root, and the rootID. If the message is considered to be valid, it will be considered for updating the rootID and sequence numbers and on correcting the time and estimating the node's clock drift.

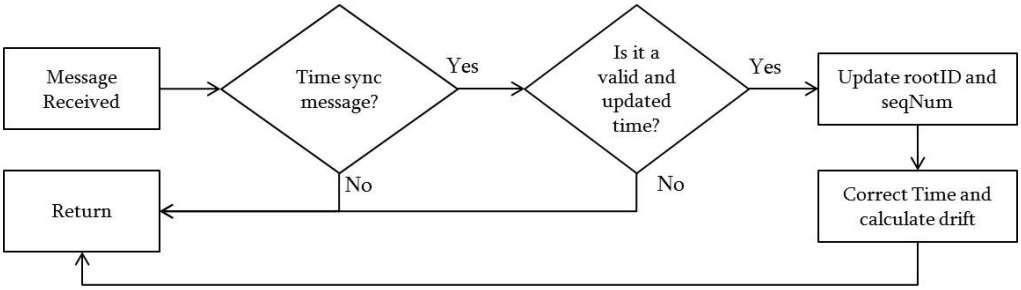

Figure 4.11: Implementation of FTSP receive function.

Upon expiration of a time synchronization period, the node reduces the "heartbeats" of the root, a variable which checks whether the root is still considered to be alive, and in case it has collected the reference points it then broadcasts the new time. These events are shown as a block diagram in the Figure 4.12.

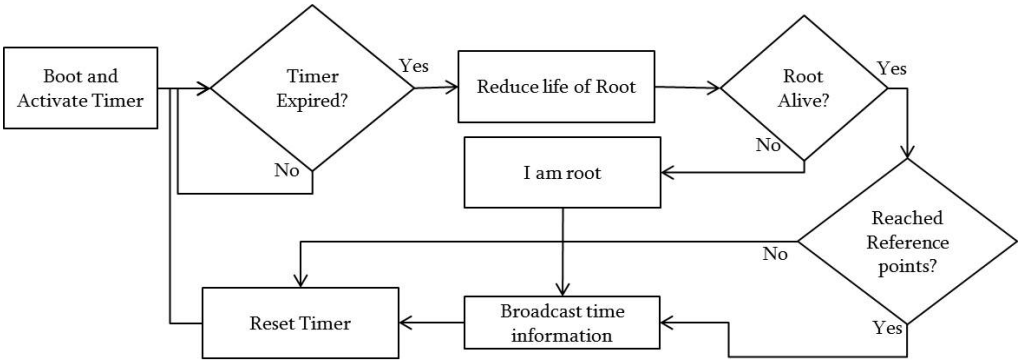

Figure 4.12: FTSP function when a synchronization period is over.

Gradient Time Synchronization Protocol (GTSP)

The main features of GTSP [18] are presented. GTSP is conceptually different from FTSP. GTSP synchronization is not based on following the time of a reference (root) node, but on the common agreement of time with neighbors. The

information included in the messages exchanged is the rate and the offset of a node. The rate indicates how quick or slow the time progresses in the node. Since the actual hardware clock rate cannot be changed, the logical clock is maintained as a software function and is only calculated when a clock reading is requested. Each node upon reception of a packet, calculates its offset compared to this neighbor and maintains this value in memory.

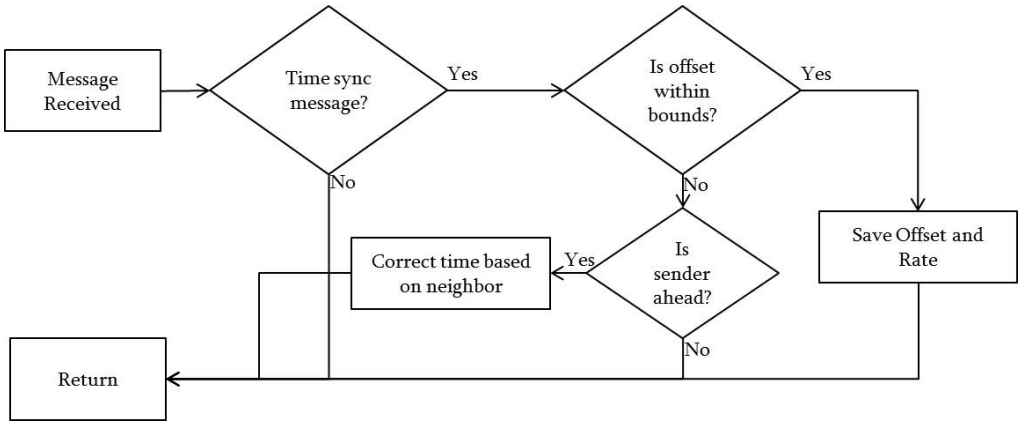

Figure 4.13: Implementation of GTSP receive function.

The full implementation of GTSP receive function is shown as a block diagram in Figure 4.13. A check on whether the offset is between some limits is made prior to using it, in order to avoid sudden changes to the network time. This is the protection mechanism against the major influence of a newly joined node after its startup. Such a node would have its hardware clock register initialized to zero, resulting in a huge offset which would affect significantly its already synchronized neighborhood. Hence, in case such a node receives a message from one of its neighbors, it will jump to their absolute value of time and rate. On the other hand, if any of the other nodes receive any of its messages before it gets synchronized, they will ignore it, assuming that by the next round it will have synchronized to their much bigger absolute value.

In Figure 4.14 the core function of GTSP, executed every synchronization period is presented. When a node's etimer expires, it performs an averaging over all received offsets between itself and its neighbors, received during the time interval from the last expiration time. This action helps to agree on a common absolute value of the clock. As a second step the node calculates an average of all the rates it received since last expiration time, so that it agrees with other nodes on a common rate by which the system's time will progress. In order to compensate for its drifting with respect to its neighbors, the node updates its rate according to the offset that was calculated. After completing these calculations, it broadcasts the absolute time of its clock counter and its rate to the neighbors.

It is significant to notice, that GTSP is totally distributed, which means

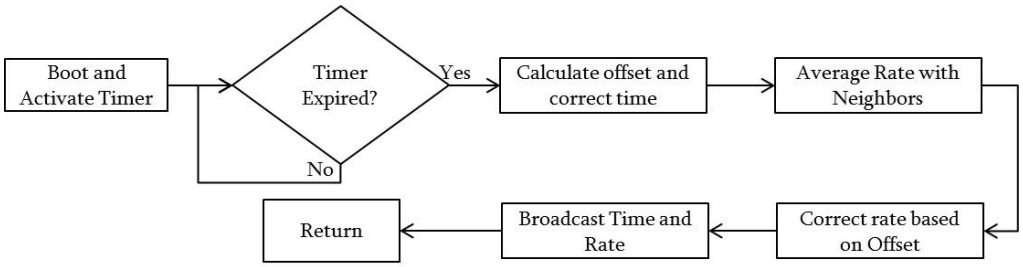

Figure 4.14: GTSP function when a synchronization period is over.

that there is not a coordinator, which would initiate the synchronization protocol. Each node instead broadcasts periodically its time and rate information, independently and not aligned to other nodes' expiration phases.

4.1.3 Timestamp External Events

The application of a time synchronization protocol implies that the nodes will be synchronized to each other after some time. The challenge to empirically measure the accuracy achieved from a protocol is addressed with the help of external events. Those events are implemented as broadcasted messages from a node, which does not belong to the network.

In Figure 4.15, the flow diagram of the timestamping and events reporting process is described. The implementation of the method to measure the performance of the protocol is designed in such a way that it is not intrusive. It runs in parallel to the normal operation of the synchronization protocol, utilizing the fact that in Contiki more processes can run in parallel.

Upon reception of a packet, the reception is timestamped and the reception callback function is initiated. In the function the message is checked whether it is recognized as a time synchronization message, based on a field of the packet structure. If it is recognized as time sync message, then all functionalities dictated by the synchronization protocol used are executed. If the packet does not include a time sync message, it is further checked whether it matches the structure of the "event" messages. In case the comparison matches, the timestamp of the reception of the message is saved.

Afterwards the process responsible for printing information about this event, which we will call "report the event" process, is started and the reception callback function completes its execution. If the message matches neither the time sync nor the event message format, it is simply ignored.

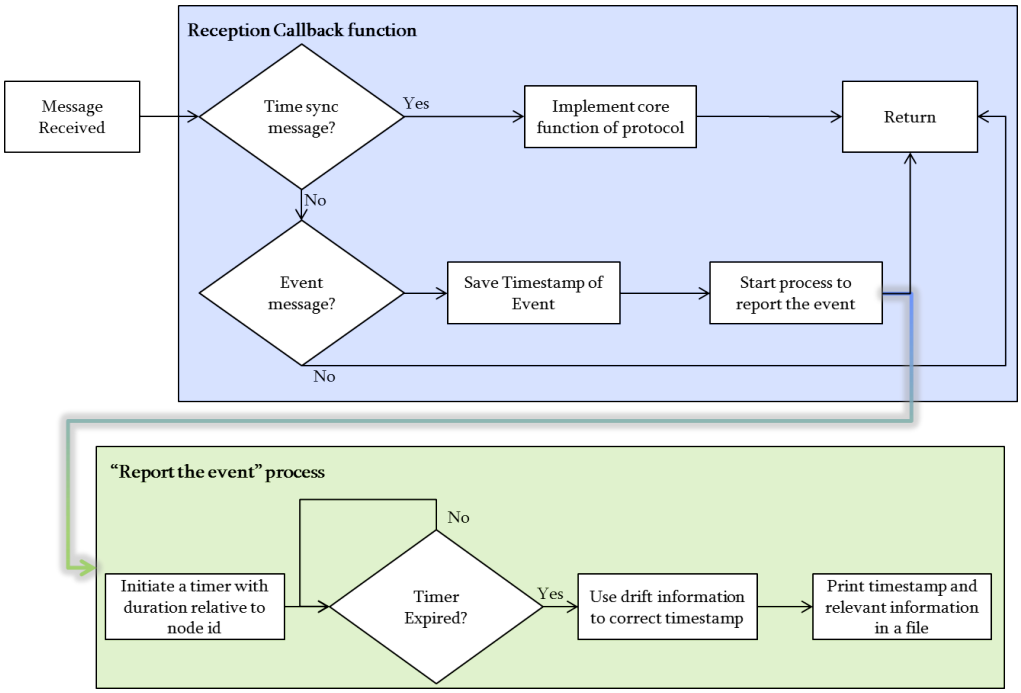

Figure 4.15: Timestamp and report of the event. The "report the event" process is called and runs in parallel to the protocol's functions.

4.1.4 Report Timestamp

The "report the event" process, as its name implies, the process is responsible to print information about the event, to report the event.

As depicted in Figure 4.15, before reporting the event, the timestamp is corrected based on the drift information calculated by the node. This is a necessary step, since time for both protocols is a software function of the hardware timer value and an offset, which results from the drift calculated by the protocols' algorithms. Upon reception of a packet the timestamp is preferred to be done as soon as possible (explained in section 4.3.2). Hence the packets is timestamps at the MAC layer, when the first bytes of the packet are valid. As the clock drifts of the nodes significantly influence the synchronization error, the drift compensation algorithms of the protocols are also used in order to correct the timestamps. More details about drift are presented in Section 4.3.1. This is a required procedure, since the protocols are designed to perform synchronization and not to report events.

After correcting the timestamp the nodes print their timestamps to a file. Since all prints are made to the same file, a TDMA approach in accessing the file is performed, in order to avoid print overlapping.

The TDMA approach is based on an implicit synchronization achieved, since all nodes start at the same point, which is the reception of the event message. The process starts with initializing a timer, with duration to expire relative to the unique node id of the node. More specifically, each node waits for duration d, calculated by:

$$d = id_{node} \cdot timeslot \tag{4.1}$$

The timeslot is to be configured based on the number of nodes that exist in the system and on the frequency the events are produced. For our experiments, the event frequency goes as high as 0.5 seconds and the number of the nodes reporting the event are up to 9. Hence in 0.5 seconds until the next event comes, 9 nodes have to report the event. In order to allow for some variance in the delay of the communication through the USB port, timeslots of 50ms are assigned. Hence the first 50ms, node with id 1 has access to the file and prints data via the USB interface. During the second timeslot, i.e. between 50ms and 100ms after the event, the node with id 2 has access to print and so on.

In such a way all the necessary information needed are gathered in order to analyze the performance of the protocol. Moreover, the collected information helps in debugging and in identifying factors that significantly influence the performance of time synchronization protocols.

An example of the information collected is shown in Figure 4.16. Each node upon reception of an event is printing one line of information which is organized in columns. All time related information is in MACA timer's units, which correspond to 4μs. Each column provides separate information:

Column:

- 1: indicates the id of the node that performs the print.
- 2: contains the print "code", a unique identifier for every print, which for events is 911-913.
- 3: contains the timestamps of the events, without correcting the timestamp according to the drift information calculated by the node.
- 4: represents the sequence number of the events produced.
- 5: for GTSP is the rate, which the node is using to compensate for the drift.
- 6: contains corrected timestamps, used for debugging purposes.
- 7: contains the corrected timestamps, after correcting them using the information about the node's drift.

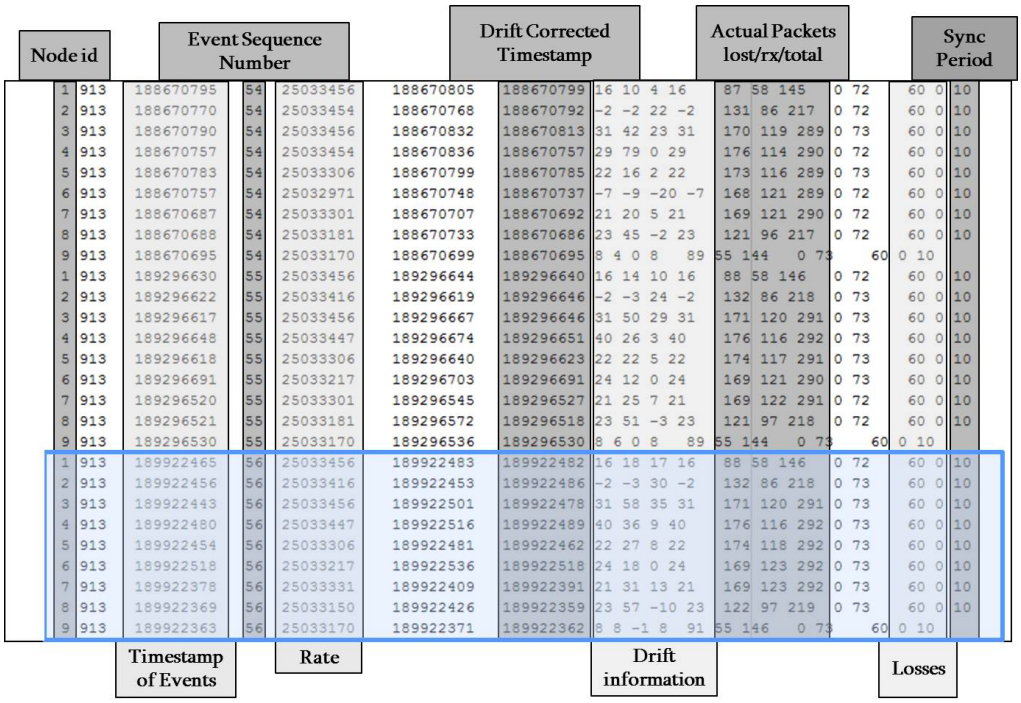

Figure 4.16: Example file including information to evaluate the performance of a time sync protocol.

- 8-11: help us identify problems, concerning the drift and the compensation used.
- 12-14: contain information about packets received and lost, information that is not allowed to be used by the node, since in reality this information is not provided.
- 15-16: provide information on the number of rounds when the node was not transmitting.
- 17-18: show the current environment parameters (packet loses or node isolation).
- 19: contains the synchronization period.

4.1.5 Data Selection

After collecting all the information in the file and ordering the events, the file contains some hundreds of thousand lines reaching a size of several MB. The text files of this size are quite difficult to be edited, since most editors cannot open the files and even if they could, it is quite time consuming to read through the

whole file for garbage prints. The necessity to provide the result files to Matlab with a specific format, which is tab separated, led to the introduction of some Linux bash scripts, to run through the file, keep only the necessary prints and clear overlapping prints.

4.1.6 Process Data in Matlab

"MATLAB is a high-level language and interactive environment for numerical computation, visualization and programming". Since a big part of the thesis required the construction of scripts to help for the analysis and visualization of the performance of the protocols, MATLAB provided a sound choice for completing this part.

Apart from the functions to complete sync evaluation tasks, some checks on the validity of the data are initially performed.

Data checks

In order to assure that the measurements taken contain valid information, some Matlab scripts are created to assure the validity of the data.

Initially the input matrix is checked to find any non-event related prints. This is done with the assistance of the second column. The matrix is run through to identify entries that do not contain a valid "code" for events, which for the setup are 911, 912 and 913.

Next checks are done for the first and the last lines of the matrix, to confirm that all nodes have completed the booting and that the test was stopped after the last event is reported.

Finally, a script checks whether all events are reported from all nodes and removes those events which are not reported from all nodes. This is necessary because the network synchronization would be evaluated otherwise for only one part of the network.

Evaluation plots

After assuring the validity of the data, the data is then used for calculations of the metrics and plotting. The framework provides flexibility in this part, allowing for many statistics and plots.

4.2 Introducing Lossy Environment

Until now the implementation has not included lossy environments. In our setup actual losses introduced are highly unlikely, since the nodes are in maximum few meters apart. Hence, in order to introduce losses, so that we can evaluate the performance of the protocol also under lossy links, two type of losses are simulated namely, random packet losses which are emulated by randomly dropping packets and packet burst losses which are emulated by blocking the transmission and reception of the nodes for a certain period of time.

4.2.1 Random Packet Losses

One common phenomenon encountered in wireless links is that the communication is unreliable. Often the links encounter low Signal to Noise Ratio (SNR), leading to high bit error rates. Most of the state-of-art technologies are using error detecting and even error correcting codes, in order to compensate for the bit errors and avoid retransmissions of packets. Unfortunately, in many applications, where primary goal is not real time response, or where a high number of sensor nodes have to be used, inexpensive hardware is chosen. This hardware usually does not support any coding techniques. In such cases the bit error rate would result in packets that are corrupted. These corrupted packets are ignored by the node, resulting in higher packet loss probabilities.

In order to simulate the packet loss, for every reception of a packet, a random number from a uniform distribution is drawn. According to the packet loss percentage that has to be simulated, a threshold is defined. This threshold is used as a comparison value for the random number, in order to get a Boolean variable, which will determine whether a packet will be considered for time synchronization, or it will be ignored. In a nutshell, this implementation simulates random packet losses with a certain probability, while the evaluation is based on tests for several probabilities of a packet getting lost.

4.2.2 Node Isolation

Another scenario observed which leads to unreliable communications in wireless sensor networks is when nodes refrain from communicating for a specific duration. This can be both a result of the node being busy, occupied by a demanding program, rebooting because of software failures or being physically obstructed by objects. Regardless of the cause, this lossy situation, where a node fails to communicate for a specific duration of time with a certain probability, is evaluated. The evaluation of the protocols' performance is done considering different isolation durations.

4.2.3 External Loss Trigger

While designing how the lossy environment parameters will be applied to the system and tested, two possible solutions look reasonable. Nodes can be either pre-programmed to experience some losses (i.e. drop some packets) after some time, or information about the lossy environment can be communicated in real time by an external loss trigger.

The first approach introduces several inconveniences, because it is more difficult to test different scenarios. For instance, applying different loss probabilities would require flashing (i.e. downloading new software) the nodes once more with different inputs. As an alternative the nodes can be programmed in such a way, that after some time interval the loss related variables are updated. However, the flashing procedure and access into the common file for writing often experiences some instabilities due to hardware problems. In such case some nodes need to be flashed again, a fact that makes it more difficult to assure a common starting point. Without such a point, the evaluation scripts cannot recognize which lossy conditions are tested and when are the nodes affected by those.

The second approach on the other hand is more realistic, since it applies the lossy environment parameters externally. Independent of the type of losses, random or node isolation, each lossy environment constitutes a lossy case and for every lossy case an external loss trigger is communicated, by means of a broadcast message, by a node which does not belong to the network. With this approach the common starting point is naturally provided, easing the evaluation afterwards. Since the number of nodes that can be used for our setup is already limited, it is decided that this information is broadcasted by the same node which is also producing the events. This eases the procedure, since the additions needed to be made are the introduction of new message types. Furthermore only one node needs to be flashed in order to change those parameters.

4.2.4 Testing Methodology

Both in the case of random packet losses and node isolation packet losses are encountered. A packet loss influence on the network synchronization depends on the link affected by the packet loss. As an example, for a tree-like synchronization protocol, a packet lost close to the root may influence more the overall network, rather than a packet lost at the last hop of the network. Moreover, the impact may be different depending on which node does not broadcast its time. For example, a node whose drift is approximately equal to the average drift of all nodes, would have a less significant impact if its broadcasted information was not received, compared to a node whose drift is the maximum among all nodes' drifts. Lastly the relative occurrence of a packet loss with respect to the synchronization interval, i.e. at the beginning of a synchronization round or at the end, influences differently the synchronization achieved.

All these parameters increase the variance in the synchronization accuracy achievable under a certain lossy environment, dictating the necessity of evaluating every scenario more than once, so that the results can be more representative.

Simple test

In this section an introduction is given into the impact of the lossy environment and the methodology followed to produce statistics about each lossy case. The section targets introducing the reader into the results section.

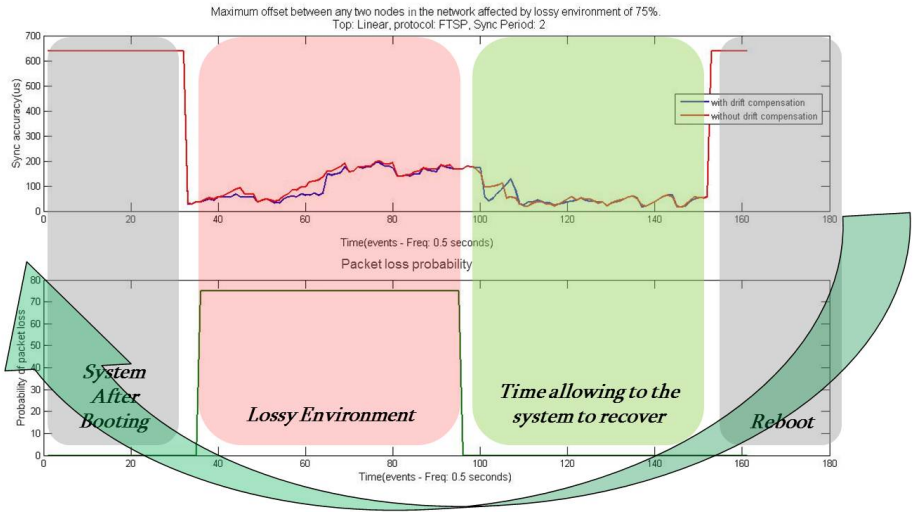

Figure 4.17: The 4 time intervals to test lossy conditions.

In Figure 4.17 a period of time is presented where a specific lossy environment is tested. The measurements are a small part of an experiment, where the topology is created by 9 nodes connected in a line, FTSP protocol is used, the synchronization period is 2 seconds and events (i.e. ask from all nodes to report their time) are produced every 0.5 seconds.

On the x-axis is time, counted into events, with frequency of 0.5 seconds. On the y-axis of the above subplot is the synchronization accuracy, which is the maximum offset between any two nodes in the network. The second subplot shows how the random packet losses are imposed into the system. In this figure, the packet loss probability is 75% for events 35 until 95 and 0% for the rest of the time.

In order to make results more readable, when the synchronization error is over the threshold of 650µs , the value is capped to 650µs.

The timeline of the figure is divided into 4 different areas. The first and fourth areas correspond to the same system state, which is the time right after a

reboot, which come before the application of a lossy phase and after the recovery of the system from the lossy phase. In other words, it is a repeating sequence of 3 areas.

The pink, second, interval is the interval where the lossy environment is evaluated. At the beginning of this area, the external node has broadcasted the values of the variables for the lossy environment to be evaluated. Nodes upon reception apply these values, resulting in lossy communication. In the interval shown in green color, the time duration for recovery is depicted during which no losses are influencing the communication of the protocol's messages. The first and fourth intervals begins with a reboot, so that the system restarts such as any residual effects are eliminated in the analysis.

There is reasoning behind the existence of each interval. Initially, before testing any environment we make sure that the previous tests do not influence the current measurements. Performing reboot is necessary, because there are many cases where the system does not converge back to a stable operation and any results gathered would include the accumulated error from previous tests. Hence the first and fourth areas are isolating and separating the test runs to have totally independent results from all test runs.

The lossy environment could be either random packet losses, or random selection of nodes to fall into isolation mode, where they do not receive or transmit anything. The statistics for this interval are collected, to help us evaluate the performance of the protocol under the specific lossy conditions.

The recovery area can provide us with statistics about whether the system could recover from a lossy case or not.

Repetition of same lossy environment

In Figure 4.18, a loss probability is applied to all nodes in the network at three different moments. However, since the losses are probabilistic, different links may be affect during each separate lossy interval.

It is clear that for all three cases the probability of packet loss influences the accuracy achieved. This observation agrees with one's first intuition, that this probabilistic loss of packet can have different outcome depending on the links that are affected. Hence for the test, in order to gather more representative results, each environment is tested several times to minimize the variance of the behavior for this lossy environment and moreover cover most of the possible system behaviors under lossy environment.

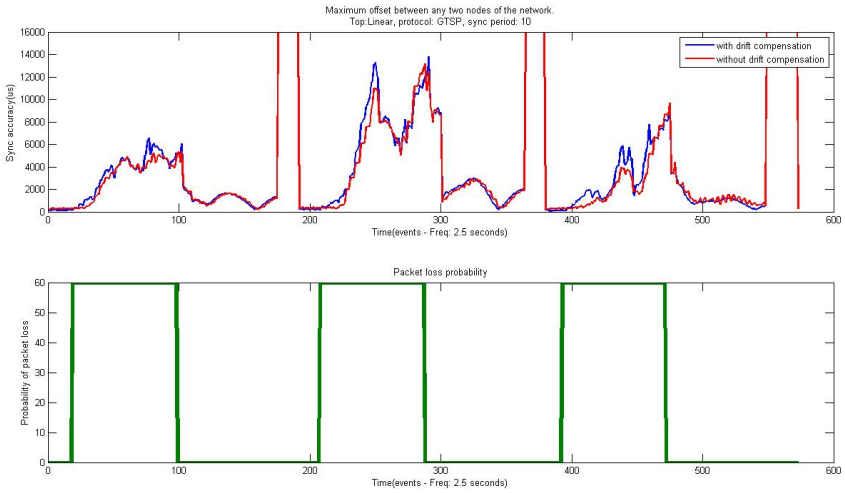

Figure 4.18: The same lossy conditions may have different influence on the total sync accuracy.

4.2.5 Extension of Event Trigger

In order to emulate the losses in our system, the node responsible for triggering events will also trigger the change of environment variables, which are used by the nodes for either dropping packets with a centain probability or block their communication with a certain probability. Exploiting once more the nature of Contiki, which allows multiple processes to run on a node, an additional timer is created, which expires every synchronization period. Upon expiration of this timer, the code responsible for the communication of the loss variables is executed. More specifically, the code running on the external node, which introduces the different environmental parameters and repeats the same parameters multiple times, can be described by the diagram in Figure 4.19:

In the diagram, initially the nodes encounter no losses. Then after a chosen number of synchronization rounds in lossy environment, the losses are set back to zero so that we let the system to recover. After a chosen number of rounds in recovery, the nodes reboot. During this rebooting, the nodes are restarting their counters and clearing all information they have about neighbors and drifts. This choice is made in order to make sure that previous executions do not influence future executions. This would be the end of a single test of a specific lossy case. Then the node checks if the configured number of repetitions of this lossy case are completed. If yes, it will increase the losses until all lossy cases are reached, or if not, it will repeat the same lossy case. The set of the configuration variables, such as the number of rounds to stay in lossy environment, are chosen for the test runs based on experimentation and they are further explained in the

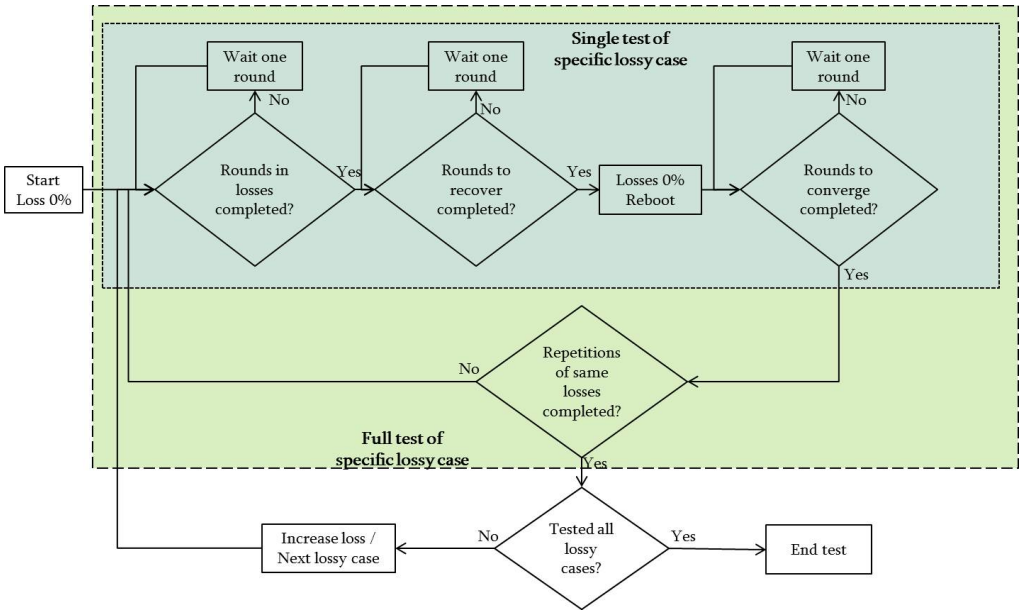

Figure 4.19: Flow diagram of code running on an external node to introduce losses.

corresponding Section 4.4 in detail.

4.2.6 Extension of Reception Routine

In order to simulate the lossy environment, upon reception of a packet some checks are done. Only if the nodes are not suffering from any losses will the checks be passed. In this case the packet is considered for the protocols' algorithms, i.e. for correcting the time and calculating the drift. These checks are introduced between the reception of the packet and the application of the synchronization protocol, as shown in Figure 4.20.

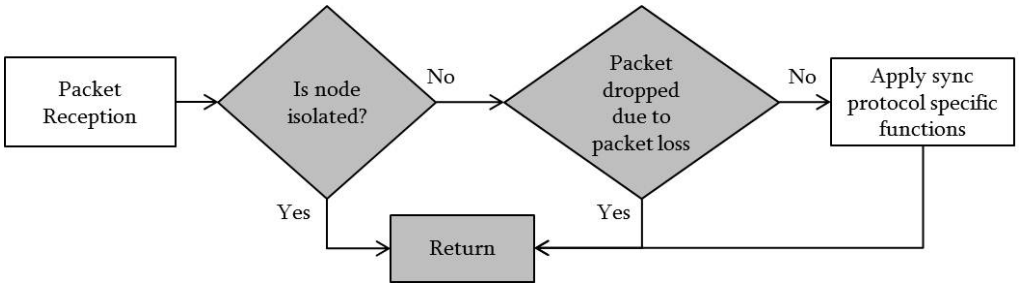

Figure 4.20: Extending receive function to support losses.

In such an implementation the framework stays non-intrusive to the functions

of the protocol, while simulating the lossy environment at the same moment.

4.3 Mitigation Techniques

In this section an analysis of factors which influence the accuracy achieved from the protocols is presented. After analyzing each influencing factor, a corresponding technique, which mitigates this factor is explained.

4.3.1 Drift

In wireless sensor networks each node has its own physical clock. This clock is an electronic circuit which is counting oscillations at a specific frequency of a quartz crystal, which for the platform used is 24MHz. A counter register is incremented per every oscillation, until a comparison value is reached, which generates an interrupt. The interrupt is implemented as a clock tick, increasing the value of a register, which functions as a timer. After the interrupt, the counter register is reinitialized and the same procedure is repeated until the next interrupt and increase of the timer's value.

In practice, the quartz crystals in each of the nodes will run at slightly different frequencies. Hence the clocks will tend to diverge from each other. This phenomenon, which causes the clocks to deviate from their nominal rate and run at slightly different speed compared to other nodes' clocks is called drift. Hence different nodes after some time "drift apart". In the literature the term "skew" has also be used to describe this phenomenon [8].

Both protocols to be tested are using some drift compensation mechanisms, to overcome this effect of the nodes' clock frequencies deviating from their nominal rates. Though, it is important to notice, that apart from the synchronization method itself, the timestamps of the events need also to be corrected, taking into consideration the drift and the time that has elapsed since the last synchronization point.

Drift estimation and compensation

The concept of drift and its compensation is formalized as follows[19].

In this scenario, there is a server, having the master clock, and a client with the task to synchronize to the server. Assuming a first order model for the master clock, it can be written:

$$x_s(t) = \varphi_s \cdot t \ + \ \theta_s, \tag{4.2}$$

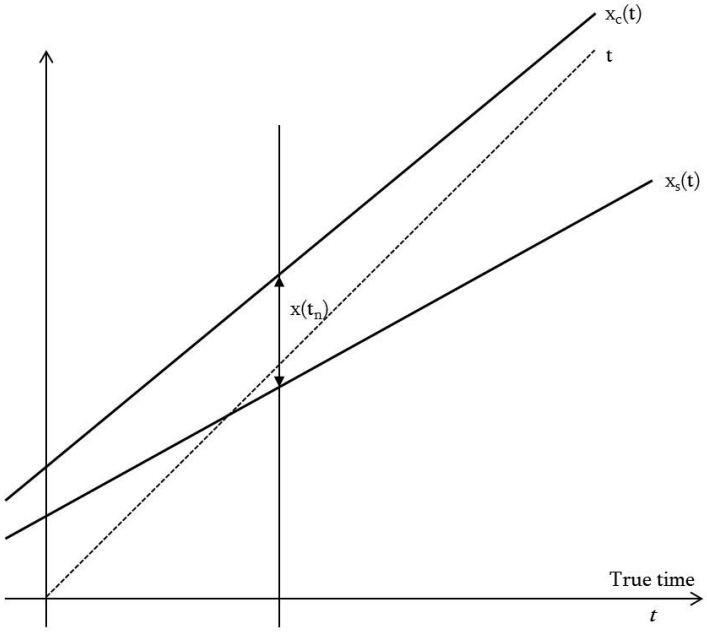

Figure 4.21: Time relationship between server clock $x_s(t)$ and client clock $x_c(t)$.

where θ_s is the time offset, φs is the clock frequency, and t denotes true time. High quality clocks have $\varphi_s \approx 1$, though in the case of the wireless sensors, the nodes tend to suffer from inaccuracies. Similarly, for a client clock (slave clock),

$$x_c(t) = \varphi_c \cdot t + \theta_c. \tag{4.3}$$

In digital systems, $x_c(t)$ and $x_s(t)$ are the values of counters at time t. The varying offset of the client $x_c(t)$ compared to the reference clock of the server $x_s(t)$ is defined as $x(t)$ and hence:

$$x_c(t) = x_s(t) + x(t). \tag{4.4}$$

In order for the client clock $x_c(t)$ to synchronize to the server clock $x_s(t)$, an estimation and compensation of the time offset $x(t)$ is necessary. the frequency deviation between the two clocks $y(t)$ is:

$$y(t) = (\varphi_c - \varphi_s)(t), \tag{4.5}$$

and the time varying offset at time t can be calculated as:

$$x(t) = \int_0^t y(t)\, dt + (\theta_c - \theta_s). \tag{4.6}$$

Based on the first order model of clock, $y(t)$ is constant, equal to y_0, simplifying the time varying offset to:

$$x(t) = y_0 \cdot t + (\theta_c - \theta_s) = y_0 \cdot t + x_0, \tag{4.7}$$

where x_0 is the absolute value of the offset between the client and the server at time $t = 0$. Thus compensation of the time offset can be improved by also taking into account the frequency offset y_0. In practice, the frequency offsets encountered are in the order of 0-20ppm.

Drift measurement

Since the drift y_0 of the nodes is one of the most influencing parameters of the network, measurements are performed in order to collect data about the nature of the drift of the nodes used in the experiments. This data is not used by the nodes in order to improve their performance, since they are collected offline with the help of a different measurement setup.

The methodology selected to measure the drift is simple enough and easy to be implemented after having implemented the FTSP. Initially a mesh topology is selected. Furthermore, node with id 1 is selected to broadcast the time once. All nodes upon reception synchronize by jumping to the time of node 1. No further messages are exchanged and events are triggered to ask for the nodes' local time.

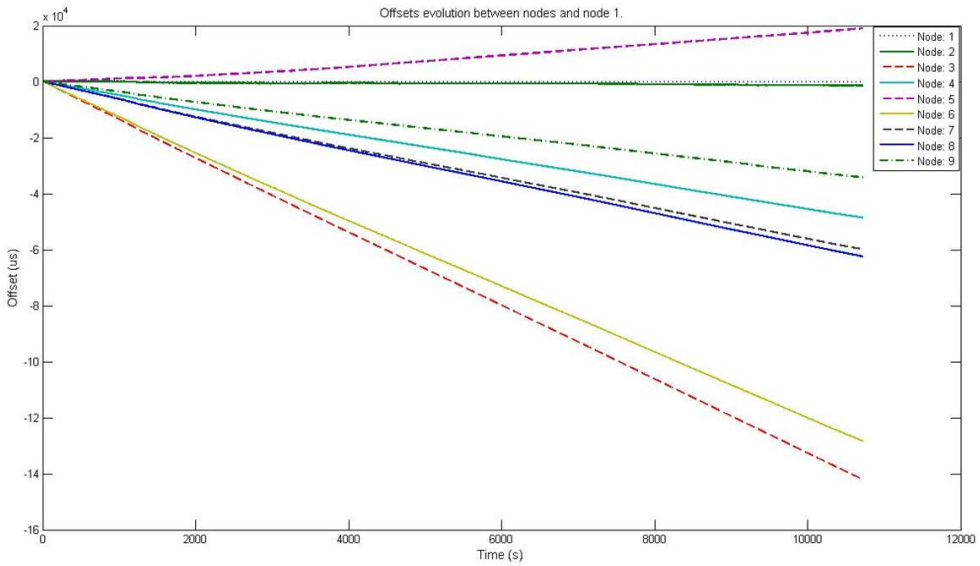

Figure 4.22: Drift plot for nodes used in experiments.

In Figure 4.22 it is shown how the offsets between the nodes and node with id 1 are evolving as time progresses. Hence the lines created show the actual drift of the nodes. It is confirmed that nodes are indeed drifting apart. Moreover the variance in the drifting for each node, meaning the change of the rate at which each node progresses its clock is the second derivative of time and is calculated to be approximately 0, allowing for linear drift compensation. The drifts for each node in comparison to the drift of node number 1 are calculated and summed up in Table 4.1.

Table 4.1: Drift measurements for nodes used in experiments

Node ID	1	2	3	4	5	6	7	8	9
Drift(µs/s)	0	-0.13	-13.26	-4.53	1.78	-11.98	-5.59	-5.83	-3.20

Drift compensation example

Since nodes diverge from each other with a stable rate, the drift compensation used in order to correct the timestamps is linearly dependent on the time duration that has passed since the last synchronization point and the drift information provided by the protocol.

As an example, for FTSP, let node with id 3 has used its algorithm and calculated an offset $\delta = -100\mu s$ from the reference time, for a setup using synchronization period T_s 10 seconds. The node will keep in memory the point of time t1, where this offset δ is calculated, from which it calculate the drift d per second:

$$d = \frac{\delta}{T_s} = \frac{-100}{10} = -\frac{10\mu\text{s}}{\text{s}}. \tag{4.8}$$

After some time, at point of time t_2, an event is broadcasted and time stamped with $TS = 1000\mu s$ by the node. At the reporting module, it will have to correct the timestamp by a value of C, where C is calculated based on the time duration $t_2 - t_1$, expressed in seconds, and the drift information:

$$C = -d \cdot (t_2 - t_1). \tag{4.9}$$

Let the time interval $t_2 - t_1$ be 3 seconds, then the timestamp TS has to be corrected by C:

$$C = -(-10) \cdot 3 = 30\mu\text{s}, \tag{4.10}$$

and the node reports the event with a corrected timestamp TS_c:

$$TS_c = TS + C = 1000 + 30 = 1030\mu\text{s}. \tag{4.11}$$

In this example, the duration $t_2 - t_1$ is expressed in units of microseconds. Though for the implementation, the total number of clock ticks during one second is used, i.e. approximately 250 000. This provides a better accuracy calculating the portion of time passed from the last synchronization point until the time where the event is broadcasted.

Time correction

The correction of a client's time based on the time broadcasted by the server can be done as follows [19].

The most basic synchronization method includes a clock server, having the reference time distributing it by broadcasting messages containing its clock reading $x_s(t_i)$. Any node client receiving this message can synchronize its clock. At time t_i, the server broadcasts a sync message with timestamp $T_s = x_s(t_i)$. This message is received by the client at client time $T_c = x_c(t_i + d_{sc})$, where dsc is the message transmission delay of the sync message. Given T_s and T_c, the client clock can compute its current offset θ_c by:

$$\theta_c = T_c - \frac{\varphi_c}{\varphi_s}(T_s - \theta_s) - \varphi_c \cdot d_{sc}. \tag{4.12}$$

Equation 4.12 is obtained by inserting t_i and $t_i + d_{sc}$ into 4.2 and 4.3 respectively and eliminating the unknown t_i. Since it is assumed that the reference time is the time that the server holds, we have $\varphi_s = 1$. Moreover, neglecting the frequency error for the message exchange, i.e. setting $\varphi_c = 1$, 4.12 simplifies to:

$$\theta_c = T_c - T_s - d_{sc}. \tag{4.13}$$

Hence, knowing the message transmission delay d_{sc}, the client can thus determine its clock offset θ_c and compensate for it. The client takes its local clock reading $x_c(t)$ and translates it into its estimate of reference time by:

$$t' = \frac{x_c(t)}{\varphi_c} - \frac{\theta_c}{\varphi_c} \approx x_c(t) - \theta_c, \tag{4.14}$$

i.e. the clock reading is corrected by the offset θ_c. Although in other systems, such as GPS-based time distribution, the d_{sc} can be obtained from position information, in wireless sensor networks d_{sc} can be determined by the two-way measurement method.

Two-way delay measurement

The basic methodology to estimate a message transmission delay over a hop, is by measuring the echo of a message and divide the round trip delay by two. Though, since the software delay from the moment of the reception of a such echo message until the reply is produce can be high enough, a more sophisticated methodology [19] is used for wireless sensor networks.

In order to more accurately estimate the message transmission delay over a hop, a method is used, where timestamps are exchanged. More specifically, the a message exchange is initiated by the client; the client timestamps an outgoing message to the server; the server timestamps its reception and responds, including the reception timestamp and an estimate of the transmission time. The client then has to timestamp the respond and has knowledge of 4 timestamps, out of which can calculate the one hop delay.

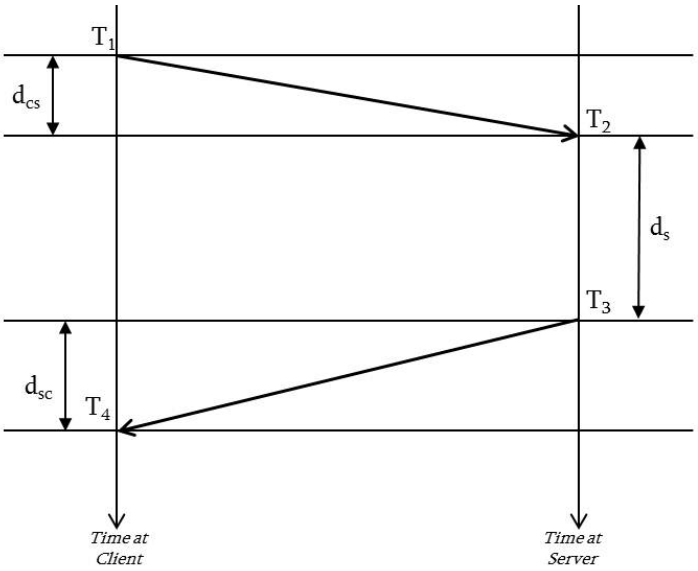

Figure 4.23: Two-way delay measurement.

This procedure is also formulated. At true time t_n, or client time $T_1 = x_c(t_n)$, the client sends a request message to the server. The message is received by the server at true time $t_n + d_{cs}$, or server time $T_2 = xs(t_n + d_{cs})$, where d_{cs} is the transmission delay from client to server. Using the linear clock models as above, we have for timestamps T_1 and T_2:

$$T_1 = x_c(t_n) = \varphi_c \cdot t_n + \theta_s, \tag{4.15}$$

$$T_2 = x_s(t_n + d_{cs}) = \varphi_s \cdot (t_n + d_{cs}) + \theta_s. \tag{4.16}$$

The server responds at time $t_n + d_{cs} + d_s$, where d_s is any processing time in the server. The response message carries the time stamps T_2 and T_3, where $T_3 = x_s(t_n + d_{cs} + d_s)$. This message is received by the client at time $t_n + d_{cs} + d_s + d_{sc}$, and the client timestamps it with $T_4 = x_c(t_n + d_{cs} + d_s + d_{sc})$,

$$T_3 = x_s(t_n + d_{cs} + d_s) = \varphi_s \cdot (t_n + d_{cs} + d_s) + \theta_s, \tag{4.17}$$

$$T_4 = x_c(t_n + d_{cs} + d_s + d_{sc}) = \varphi_c \cdot (t_n + d_{cs} + d_s + d_{sc}) + \theta_c. \tag{4.18}$$

The 4 timestamps T_1, T_2, T_3, T_4 are now available at the client. Using those the client can calculate the unknown delay $(d_{cs} + d_{sc})$ and its clock offset θ_c by solving 4.15 to 4.18, yielding

$$d = \frac{T_4 - T_1}{2\varphi_c} - \frac{T_3 - T_2}{2\varphi_s}, \tag{4.19}$$

$$\theta_c - \frac{\varphi_c}{\varphi_s}\theta_s = \frac{1}{2}(T_1 - \frac{\varphi_c}{\varphi_s}T_2) + \frac{1}{2}(T_4 - \frac{\varphi_c}{\varphi_s}T_3) + \varphi_c\frac{\Delta d}{2}, \tag{4.20}$$

where $d = (d_{cs} + d_{sc})/2$ is the average one-way delay and $\Delta d = d_{cs} - d_{sc}$ is the asymmetry in the bidirectional delays, i.e. $d_{cs} = d + \Delta d/2$ and $d_{sc} = d - \Delta d/2$.

The assumptions made in order to simplify the formulas are:

- delays are symmetric, i.e. $\Delta d = 0$,
- the clock server provides the reference clock, hence $\varphi_s = 1$ and $\theta_s = 0$,
- the client frequency error is neglected during this message exchange, i.e. $\varphi_c = 1$.

The formulas then simplify to:

$$d = \frac{T_4 - T_1}{2} - \frac{T_3 - T_2}{2}, \tag{4.21}$$

$$\theta_c = \frac{T_1 - T_2}{2} + \frac{T_4 - T_3}{2}. \tag{4.22}$$

The client then corrects its time $x_c(t)$ based on the estimated offset θ_c. This method provides estimation of the one hop delay and synchronization accuracy in scenarios where the communication between client and server can be assumed stable. Any variance in the one hop delay, or asymmetry in the behavior of the

two communicating parts, deteriorates the estimates of d and hence the quality of the synchronization achievable. However, in cases where variance is indeed existing, the common strategy to deal with such variance is to keep track of those values, by performing measurements for more than one time points and apply a more sophisticated method to estimate the one hop delay.

4.3.2 MAC Layer Timestamping

MAC layer timestamping was first introduced in [5], a technique which aims in reducing the delays between the transmission and reception of a time synchronization message. Although this technique would be not only state of art, but also required method to increase the performance of any protocol which includes time information in the messages, it is important to point out how it helps to reduce delays introduced by the operating systems and the medium access.

The synchronization protocol is implemented as a program, running on top of an operating system. Most of the current platforms support multiple threads running on the central processing unit. In most industrial applications, many applications besides the synchronization protocol occupy the CPU, with different priorities. In all cases where the synchronization program is assigned lower priority from the system designer, it may be interrupted from other processes with higher priorities. If the timestamping was performed at the higher, usually called application layer, of an hierarchy of network layer, it may be that the process executing the synchronization, after saving time information to be communicated to neighbors, gets interrupted from other processes of higher priority. This would result in outdated time information, decreasing significantly the overall accuracy of the system.

As far as the medium access delays are concerned, they are mainly an effect of multiple access schemes used. Typically in the wireless medium with restricted bandwidth, TDMA-based and/or CSMA-based schemes are used. Although it is not of interest to analyze the latency introduced by those schemes, it is a fact that nodes would have to wait a variable and significant amount of time, until they are allowed to access the medium and transmit. Once again this delay can occur between the timestamping and the real sending of the message, biasing the time information and degrading the accuracy of the system.

The method of timestamping at the MAC layer provides the necessary countermeasure to deal against those delays. Time synchronization packets need to be timestamped before their transmission and their reception. This timestamps are readings of counters. The point of time at which those counters are read is chosen at the MAC layer, assuring that the delay left in the one hop delay in negligible, or at least deterministic. This reduction of the variance is in reality enough, to achieve the same performance as if the delay is negligible.

In Figure 4.3.2[1] the solution provided by the MAC layer timestamping is explained in a simple example.

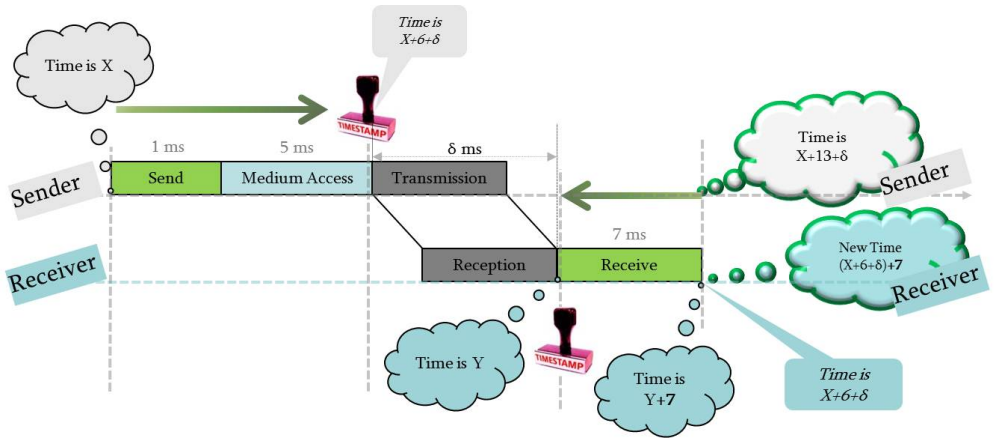

Figure 4.24: Timestamps at the MAC layer reduce the variance of the delay per hop.

This is a typical scenario where the sender wants to send some time information about its local time and the receiver has to synchronize to this time. The message exchange is shown, from the moment that the application layer gives the command for the packet to be sent, until the receiver reads the value in the packet.

The sender, after executing the synchronization protocol wants to broadcast time information. The command "send" is issued at time X. Though the command takes say 1ms until it is executed, because the CPU was busy by another process. Then, when trying to acquire the medium, the node realizes that there is an ongoing transmission, according to CSMA/CA it backs-off, tries again but still sees an ongoing transmission until after say 5ms manages to gain access to the channel.

Timestamping is performed right before the actual transmission of the packet. Moreover assuming that the node knows the delay δ ms from the moment it transmits the packet until the time information is read from the receiver, it can include this value into the time field of the packet, which will at the end sum up to the value $X + 1 + 5 + \delta = X + 6 + \delta$ ms.

Upon reception, the receiver timestamps the incoming packet, at moment Y. The CPU is busy, hence the application running the synchronization of the node has to wait for 7ms. This delay though can be measured, since the node has timestamped the message upon reception. Hence, when the synchronization application is called to correct the receivers time, it read $X + 6 + \delta$ ms in the

[1] based on lecture notes by Prof. Dr. R. Wattenhofer, "Ad Hoc and Sensor Networks", lecture "Clock Synchronization",2010, ETHz.

packet, adds the 7ms of delay to correct its local time to $X + 13 + \delta$ ms, the same with the sender.

Since the accuracy of measuring the OS and medium access delays are in the range of few microseconds, these delays can be addressed and the synchronization error left in this message exchange is mainly a result of the uncertainty left in the one hop delay δ, uncertainty measured in the next section.

4.3.3 Hop Delay Compensation

Limitations are imposed from the choice of the platform and the operating system running on it. For both FTSP and GTSP, the authors were able to achieve timestamping taking into account the transmission and reception delays, the byte alignment delays, reducing the variance of the hop delay to the variance of the propagation delay, which is negligible, since it is in the range of nanoseconds. For the setup used for experiments, which uses a combination of Contiki OS and Redwire Econotag, the timestamping during a packet transmission is done in the function tx_packet, right before the interrupts for the MACA module are disabled. This is the last moment, at which the MACA counter can be read and after the packet is already prepared. The counter reading, i.e. the timestamp, is saved at this moment. The corresponding bytes of the transmission buffer are then modified so that they represent the last known MACA counter value saved. The transmission is then initiated. The complete interrupt sequence during transmission can be found in [20].

Thus for this platform the delay between the transmission and reception exceeds the non-negligible order of nanoseconds. In order to compensate this, measurements for the delay between the transmission and reception timestamps are realized on the platform used. Although the mean delay of the message exchange can be compensated for, the variance of these delays cannot, introducing an additional uncertainty, to the one resulting by the timer resolution described in Section 3.3.4 per message exchange and hence per hop.

There are two ways to measure this delay. The typical methodology includes two nodes exchanging two messages, a delay request and delay response, which are timestamped upon transmission and reception, as explained in Section 4.3.1. The node requesting the delay measurement has after this message exchange the information required to calculate the delay which occurs between the transmission and reception of a packet. Measurements for the hop delay with this methodology are realized and presented in 4.25. The mean value of the hop delay is 443.13µs and its standard deviation 1.78µs.

However, in order to claim that the results of this methodology are extendable for all links, some assumptions have to be made. Aside the assumption of symmetric links, the test considers only two specific nodes. The two time synchronization to be evaluated do not include this technique to deal with this

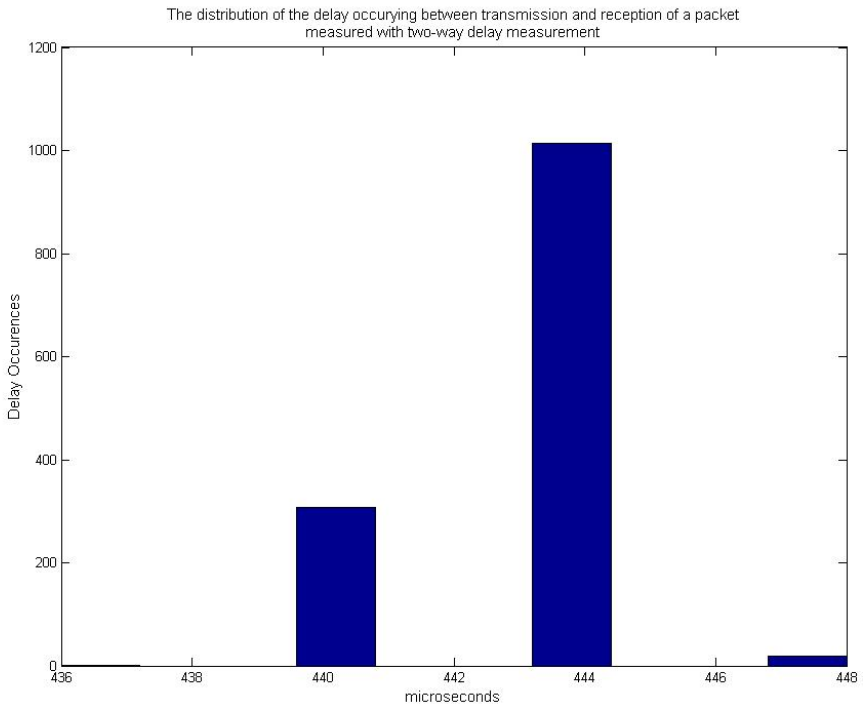

Figure 4.25: Distribution of delay between the transmission and reception of the message, measured by the two-way delay measurement technique.

delay, hence this value has to be known in advance, so that it can be compensated for. Thus in order to claim that this measurements are representative for all nodes in the network, the same technical characteristics for all nodes have to be assumed. This testing methodology also suffers from the limited resolution and in fact influences the accuracy of the four timestamps exchanged. Lastly rounding errors limit the quality of such a measurement technique.

A second methodology is also implemented, whose setup is explained. Mesh topology is created and a special condition is chosen; only node number 1 is allowed to transmit. Hence the nodes will all receive the time from node with id 1 and they will correct their time accordingly. Moreover, events are broadcasted every 0.5 seconds, while the synchronization period is 4 seconds. In order to avoid, as much as possible, the effect of the clock drifting, only the measurements belonging to the first 0.5 seconds are taken into account. Technical reasons do not allow for higher frequency of producing events, because the reporting process, which follows a TDMA schedule based on the ID of the nodes, requires 400ms plus the time needed for printing.

The differences of the reported time between the node number 1 and the rest

nodes are calculated and the results of the measurements for the 8 nodes are included in the Table 4.2:

Table 4.2: Differences of reported time between node 1 and nodes 2-9 (mean and standard deviation).

Node id	2	3	4	5	6	7	8	9
Mean(µs)	446.60	449.77	447.81	446.60	449.30	448.00	448.00	447.07
Std(µs)	2.60	2.93	3.49	3.37	3.57	3.38	3.02	3.00

The average difference over all measurements is 447.90µs and the standard deviation 3.34µs. In the next plot, the distribution of these differences is shown in Figure 4.26.

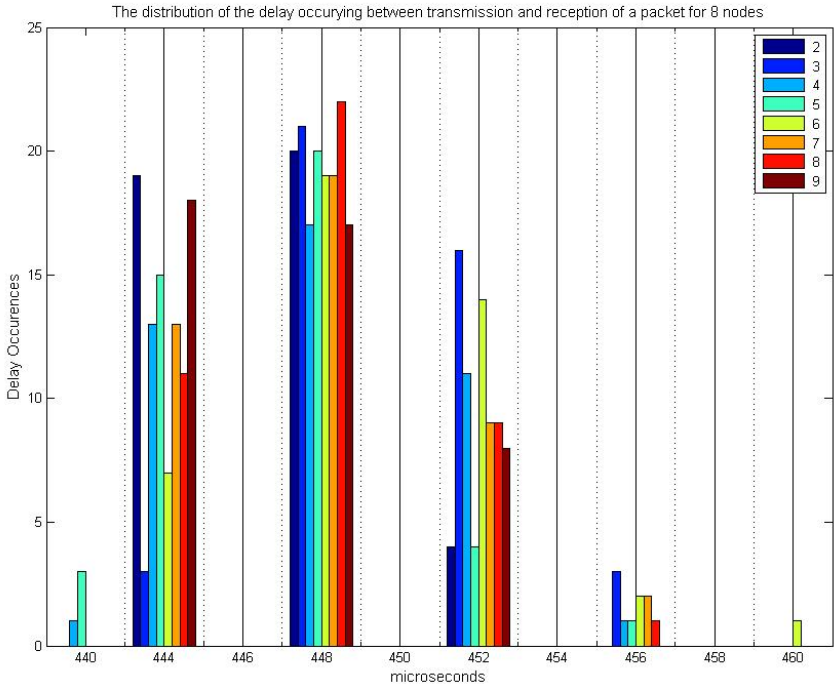

Figure 4.26: Distribution of delay between transmission and reception of a packet for all nodes.

Both in Figure 4.26 and in Table 4.2, the slower nodes, e.g. node 3 and 6, result in higher differences . Thus the drift has still an important influence on the measurements. This influence does not allow to make safe conclusions for the one hop delay. However, using the measurements of Table 4.1 for time duration 0.25 seconds, since the exact time duration cannot be known, the corrected measurements, which are used for the one hop delay are gathered in Table 4.3.

Table 4.3: One hop delay (mean and standard deviation).

Node id	2	3	4	5	6	7	8	9
Mean(µs)	446.57	446.45	446.68	446.16	446.31	446.60	446.54	446.27
Std(µs)	2.60	2.93	3.49	3.37	3.57	3.38	3.02	3.00

Clearly the standard deviation remains the same for the measurements, because the corrected delay is derived from the measured delay and the addition (or subtraction) of the measured drift for each node.

The average one hop delay over all measurements is 446.45µs and the standard deviation 3.16µs. The overall distribution approaches a normal distribution, since calculations show that 63.7% of the values belong to the 2σ, 96.5% belongs to 4σ and 99.7% to 6σ area around the average value, compared to 68%, 95% and 99% respectively. Such an observation allows to assume that no systematic errors are introduced to the measuring technique.

This measurements are important for the implementation of both GTSP and FTSP, in order to mitigate this delay. More specifically every time a node has to report its timestamp, it should correct its time by 446.45µs on average. Of course since the resolution is limited to 4µs, the implementation includes the either the value 444 or 448, with 39% and 61% probability respectively, at every transmission, leading to an average correction of 446.44µs.

4.3.4 GTSP Time Average Resolution

While using GTSP protocol, the averaging of time offsets and rates in cases where tight synchronization is already achieved, another technical problem arises that has to be solved. GTSP protocol uses the averaging over the time differences between nodes and neighbors. The restriction of the 4µs of time resolution introduces once more an error in combination with the rounding error resulting from the division.

In order to mitigate this error, the rest of the division is saved and used for the next time corrections. The same implementation is also used for the averaging of the rates. Since the values of previous synchronization rounds are used in next rounds, this feature is called as “memory correction” in Figure 4.27. This can improve the performance of the synchronization algorithm, especially in the cases where the system has reach a stable state.

The results of Figure 4.27 concern a mesh network, of 9 nodes, using GTSP and synchronization period of 5 seconds and event generation every 1.2 seconds. The y-axis shows the synchronization accuracy of the network, in means of global synchronization offset, i.e. the maximum offset between any two nodes in the network, while the x-axis represent the different time instances, i.e. broadcast

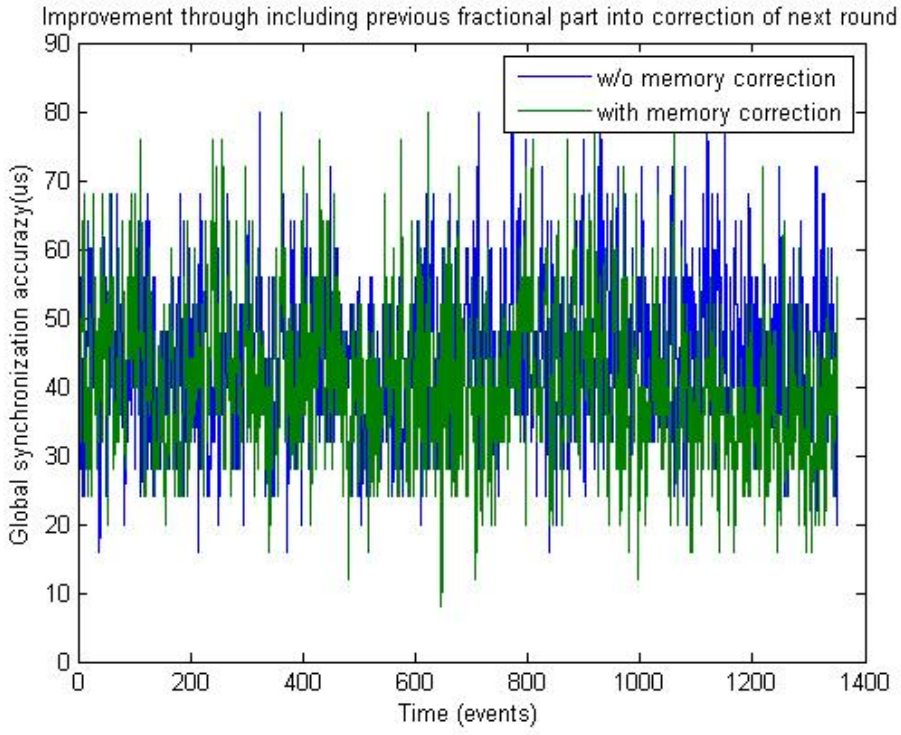

Figure 4.27: GTSP memory correction feature improves the performance in stable state.

events. The mean of the green line appears to be a bit lower than the mean of the blue line, achieving slightly better synchronization accuracy. On average, without the using this extra information saved from previous synchronizations, the system achieves on average 44.2µs accuracy, while the use of the fractional part from previous executions improves the performance leading to 40.6µs, an improvement of 8.1%. Especially in the last part, where the system appears to be quite stable, the performance with the use of the fractional part achieves synchronization of 36.6µs for the last 350 events, while without this improvement the synchronization is at 45.3µs, consisting an improvement of almost 20%.

Similar results are observed in more experiments. Both intuitively and by experimentation it is concluded to use this small optimization step. Although the improvement may be considered unimportant, it shows how technical details can also influence the achievable accuracy.

4.3.5 GTSP under Losses

In the diagram of Figure 4.28, the GTSP algorithm includes the steps, which do not have a colored background. These begin with the timer, whose expiration

triggers the end of a synchronization period. The average offset between the node and its neighbors is then calculated, and based on that value its offset is corrected. Afterwards the rate is averaged with the received values of its neighbors. The rate of the node is corrected, taking into account the offset from the average time that is calculated. Finally the node is broadcasting its time and its rate.

Tracking rounds alone

In a lossy environment, messages are lost and the time differences between nodes get bigger, since they accumulate for every synchronization period that the nodes do not synchronize. After some synchronization periods of loss, the nodes might be able to receive a neighbor and the measured offset does not correspond to the time interval of one synchronization period. In such a case, the influence of the offset on the rate is not as expected, leading to the divergence of the algorithm. Although there is no clear strategy to completely avoid such results, the influence of such cases can be addressed to a certain degree.

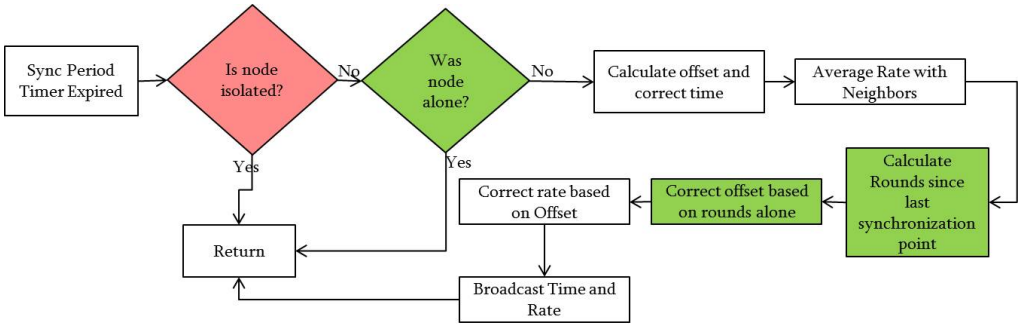

Figure 4.28: GTSP tracking the rounds that the node was alone.

Initially a simple example is analyzed, to make the nature of the problem more clear. Let two nodes drifting away by 2ms per second; the quick node has a rate of 1.001s and the slow 0.999s, they both believe that they have a rate of 1.000s. Hence in one synchronization period, the reported offset would be 2ms. If time information is exchanged, both nodes will correct their rates, so that the slow node keeps in memory a rate of 1.001s and the quick node 0.999s, compensating for the real clock drift. If however the packets are lost for say four synchronization periods, then the offset after the fifth synchronization period would be 10ms. If these 10ms were used for correcting the rate, which represents the speed that a node progresses in one second, then the quick node would decide its next rate to be 0.995s and the slow 1.005s. Such a big adjustment is not only not correct, but puts also the convergence of the protocol at risk. At this point it should be noticed that this is a simplified example, serving only the purpose

of explaining the problem. In practice more factors need to be addressed, such as the not aligned expiration of the two nodes.

In order to mitigate this effect, nodes are keeping in memory the last time that they had a synchronization with at least one neighbor. Hence when an offset is calculated, it is firstly translated into the offset that corresponds to the time interval of one synchronization period by simple division. Notice that this approach does not completely solve the problem, but helps to mitigate the quick divergence of GTSP.

The implementation of this feature includes the introduction of new checks, which can be seen in Figure 4.28 in green color. With red is the implementation of the nodes' isolation lossy environment.

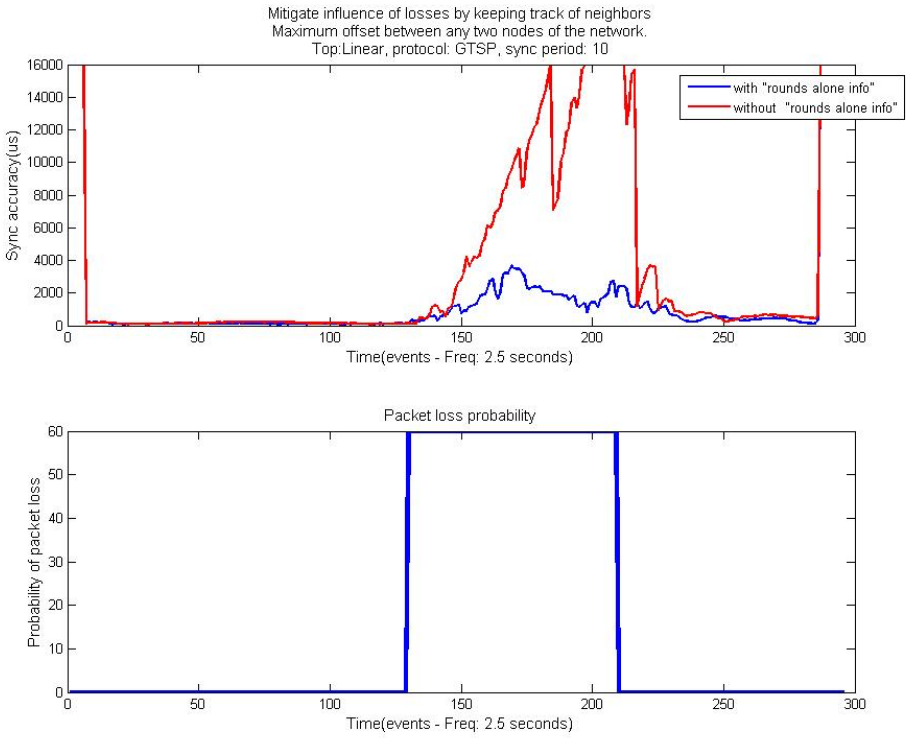

Figure 4.29: Improvement achieved by the "rounds alone" feature.

In order to evaluate the introduction of this loss-resistant feature, two separate experiments are performed. The setup for both experiments is the same: random packet loss probability is 60%, topology is linear, sync period is 10 seconds. In Figure 4.29, the improvement achieved by this introduction is presented, where on the top plot, the global synchronization offset is presented, increasing under the impact of the lossy environment, which is shown in the lower plot.

This may be an extreme case comparison, where in the first case the system

went quickly to an unstable case. Though through all measurements performed, repeating the same scenario 25 times, all times the proposed counter measure outperforms the simple GTSP, reducing the average global synchronization offset by 19% and its standard deviation by 20%.

Offset limit reduction

Although with the introduction of the tracking of the rounds that a node has not received a message, GTSP still performs much worse than FTSP, being able to maintain global synchronization accuracy in the range of 1-4ms. In order to improve furthermore the performance of GTSP, the protection mechanism described by the authors, against new nodes joining the network, comes to be of benefit. This mechanism makes sure that nodes which detect an offset larger than a threshold, they should jump to the neighbors clock value (see Figure 4.13). In order for this mechanism to be useful when the system is under losses, the threshold of this value should be configured according to the expected drift of the nodes. Initially the threshold was set at 1ms, which in cases of normal loss free environment would be considered way too small, based on the fact that during startup the clock offsets between nodes are in orders of seconds or even minutes. This feature is exploited by varying the threshold according to the sync period. The threshold is then set to tolerate drifts up to 20µs per second. Hence, for the 10second experiment shown before, the threshold is reduced from 1ms to 0.2ms. The two choices of the drift limit are then brought into comparison, to realize that this configuration modification can provide once again an improvement, maintaining the accuracy below 2ms.

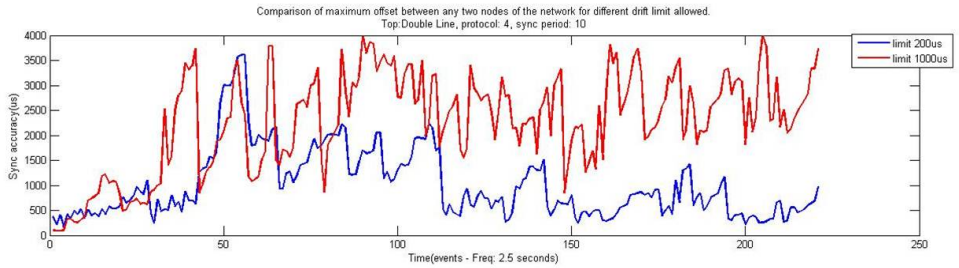

Figure 4.30: Impact of offset limit configuration.

In Figure 4.30 with the blue line is the global synchronization achieved for offset limit 200µs, while the red line is the accuracy achieved with offset limit of 1ms. In both cases the network is suffering from 60% random packet loss. On average over several experiments, GTSP with offset limit of 200µs improves the global synchronization, limiting the offset to 44% of the value with offset limit of 1ms, pointing out the significance of this configuration parameter in the overall performance of the protocol in lossy environment.

4.4 Network and Evaluation Framework Configuration

One of the strengths of the framework is that it provides flexibility in evaluating several different scenarios. The user can configure the network parameters and the parameters of the extensions which support the lossy conditions. The set of the network and framework configuration variables are shown in Figure 4.31:

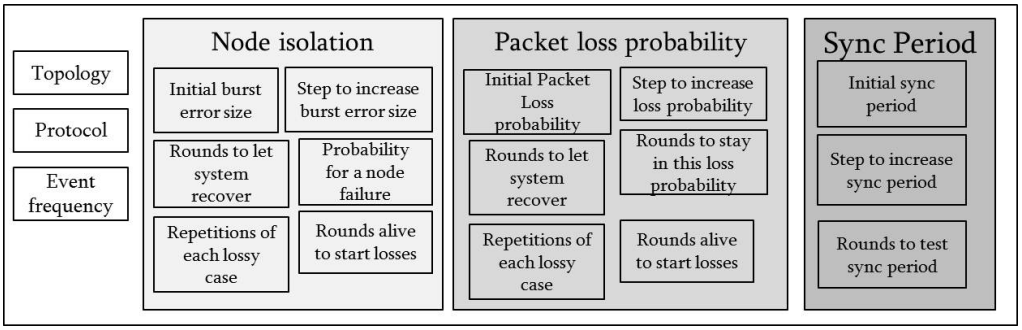

Figure 4.31: Network and framework configuration variables.

Each parameter is explained:

- Topology: The topology is created by selecting which links to be activated. Mesh, linear and double line topologies are already implemented as functions, but the user is free to implement and test any other topology.
- Protocol to be used: The framework can be used to evaluate any time synchronization protocol. The current implementation supports the evaluation of FTSP and GTSP.
- Event period: This variable configures the frequency at which all nodes will be asked to timestamp an event. Since the test configuration supports variable synchronization period, the choice to keep track of a constant number of events per synchronization period is also supported.
- Packet loss probability: The framework provides much flexibility in this variable too. The user can choose whether a static or increasing packet loss probability is preferred. In the second case, one can choose the number of synchronization rounds to evaluate a specific loss probability, to allow the system to recover in stable no-loss environment and to allow the system to converge after a reboot before the losses start. Moreover the number of repetitions of a specific packet loss probability and the step to increase this probability are configurable. Finally this loss probability, static or increasing, can be chosen to affect a specific link, or all links in the network.

- Synchronization period: The choice of the synchronization period can be done statically, which means that the user sets one synchronization period for the scenario to be evaluated. It is also possible to test several synchronization periods for a single test run. In this case, the user can choose an initial value for the synchronization period, the time duration that this period will be evaluated and the step by which the synchronization period will be increased when the previous synchronization period is considered fully tested.

- Node isolation duration: This parameter can evaluate how the network synchronization is affected by nodes which are refraining from communicating for a duration of time. This duration can once again be chosen static or increasing. The probability that a node will be in isolation, the duration to recover after a lossy phase, the duration to allow the system to converge after a reboot are also configurable. Once again, the user can choose the number of repetitions of each node isolation duration, the step by which the isolation duration is increased and whether or not all nodes will be affected by this parameter.

4.5 Evaluation Metrics

The evaluation of the performance of the protocols can be done based on several metrics. The framework supports the evaluation based on all these metrics, which can be used depending on the requirements of the target application.

The following metrics are considered, which are the output of the part of framework dealing with the data analysis. The metrics can be plotted so that their evolution as time progresses as well as their distribution can be analyzed. Moreover their average value over the whole experiment and its standard deviation are calculated, reducing the evaluation to a single set of numbers.

- Global synchronization offset

  The global synchronization often is considered as the most important metric for evaluating the performance of a time synchronization protocol in a network. For the global synchronization, the maximum absolute value of all pairwise offsets of all nodes is calculated for every event.

- Global average pairwise offset

  Also often used to evaluate a protocol is the mean pairwise offset. For this metric, the pairwise offsets of all nodes of the network are averaged.

- Local synchronization offset

For other kind of applications, it is important to evaluate the performance of a protocol considering its local synchronization. For a such evaluation the maximum offsets between any two neighboring nodes are calculated.

- Local average pairwise offset

  Similar to the case of the global synchronization, the average pairwise offset is calculated. Though in this case the offsets examined are only those between any neighbors.

- Percentage of time during which the network is considered synchronized

For all the aforementioned metrics, the percentage of time during which the network is considered as synchronized can be calculated. In this direction threshold needs to be defined, which is a configurable parameter of the Matlab data analysis. Only when the synchronization offset of the network is below this threshold, the network is considered to be synchronized. The evaluation is based on the offsets collected, when the network is synchronized.

In practical applications one goal of a time synchronization protocol could be to assure that the nodes' synchronization accuracy, measured by one of the aforementioned metrics, will be maintained under some threshold, over which the nodes are assumed to be unsynchronized. In such a way a threshold is defined, which provides further ground for results analysis. With the help of this threshold we calculate the percentage of the time, when the network's synchronization remains below this threshold.

Since the framework support the evaluation of the performance of time synchronization protocols under lossy conditions, the metrics are also compared on the basis of different lossy conditions.

Lastly several plots and statistics can be provided for further in depth analysis. In such analysis plots about the offsets per node, offsets per hop, offsets of all nodes can be provided to show their distribution and their evolution during the whole experiment.

All these metrics and statistics can be used as an input for comparison of different test scenarios. Different protocols, synchronization periods, topologies and configuration variable can be compared in order to assist a network designer with the choice of appropriate protocol and system parameters, so that the network's synchronization performance meets the respective requirements.

Chapter 5

Results

The result section demonstrates the framework at work. It should be clearly stated that its goal is neither to achieve the best possible performance nor to judge in favor or against of one of the protocols, or parameters tested, though much effort and several tests are realized in order to assure the proper implementation of the protocols' synchronization mechanisms.

The flexibility of the framework is derived from the fact that it allows testing of several configurations and it results in a multidimensional comparison of the test results. Due to time limitation, not all possible combination of comparisons are made. Though for the sake of completeness, each parameter of a category will be evaluated in comparison with a parameter of the same category.

The different configuration categories can be summed up in Table 5.1.

Table 5.1: Configuration categories to be tested.

Protocol	Sync Period	Loss Type	Topology
GTSP	1 second	None	Mesh
FTSP	10 seconds	Random Packet Loss	Linear
		Random Node Isolation	Double Line

5.1 Loss Free Environment

In order to introduce the result section, 4 simple cases (Table 5.2) are presented. The loss free environments offer for a better understanding of the performance of a protocol, while comparing the mesh and linear topology, for two synchronization periods.

Table 5.2: Example scenarios evaluated.

Scenario	Protocol	Topology	Loss Type	Sync Period
Example 1	GTSP	Mesh	None	1 & 10 seconds
Example 2	FTSP	Mesh	None	1 & 10 seconds
Example 3	GTSP	Linear	None	1 & 10 seconds
Example 4	FTSP	Linear	None	1 & 10 seconds

Running a test in an environment that does not suffer from any losses should lead to a quite tight synchronization. At the same time reducing the synchronization period, will reduce the influence of the drift and any drift compensation algorithm that is applied.

Reducing the synchronization period should not be translated to a linear reduction of the synchronization error. The reason behind this statement is the unavoidable rounding errors, the error resulting from the limited time resolution and the errors resulting from the variance of the one hop delay. Such errors can be in fact more obvious when the synchronization period is smaller, since they dominate the error resulting from clock drifts, setting a lower bound on the minimum synchronization error achievable.

5.1.1 Example Results

A simple test scenario is realized, which provides some first insight on the protocol's performance. The topology chosen is mesh, the synchronization frequency is chosen to be high, i.e. 1 second. The performance of the GTSP protocol is evaluated examining the global synchronization metric, in terms of maximum offset between any two nodes of the network, and how it evolves over time. Furthermore the distribution of the occurrences of each offset is plotted and the rate evolution is shown, to see how the protocol behaves in case of slower and quicker nodes of the network.

Global synchronization offset evolution

In Figure 5.1 the global synchronization is evaluated for this scenario.

The maximum offset between any two nodes of the network is on average 21µs, while in the same time the standard deviation is limited to 6µs, allowing to assume that the network performance is close to the best achievable. In this plot, the limitation of 4µs accuracy that derives from the 250kHz clock is observable. The maximum offset between any two nodes has a minimum value of 4µs, while next minimum are the 8µs and so on.

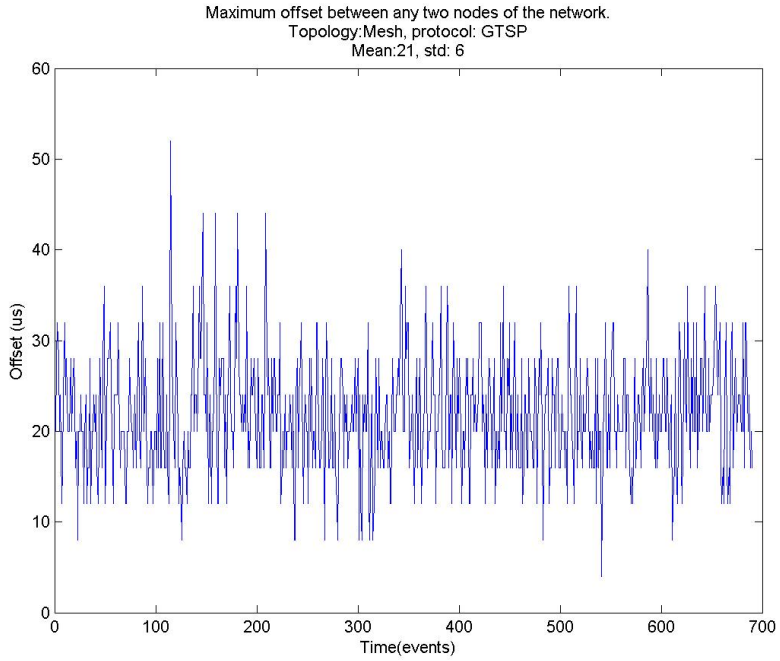

Figure 5.1: Global synchronization offset for GTSP in a mesh topology with synchronization period 1 second.

Global synchronization offset distribution

Furthermore the same results are gathered and plotted as a distribution.

In Figure 5.2 the distribution of the occurrences of the maximum offsets is plotted. The distribution approaches a Gaussian distribution. In fact in all tests performed the error could be modeled by a Gaussian distribution, confirming our analysis and ensuring that no discrete systematic errors are introduced in the system.

Rate evolution

Initially the rate evolution for all nodes around the initial value is shown in Figure 5.3:

The evolution of the rates indicates the rate by which the nodes are increasing their clock, for all nodes tested. Although all the nodes follow the similar rates, the node with id 6 depicted in light green color reports quicker rate, while node 5 with the purple reports the lowest rate. Given the figure of the drift of the nodes tested (see Figure 4.22), it can be confirmed that the nodes realize correctly their drift, since node number 5 and 6 are shown to be the quickest and second

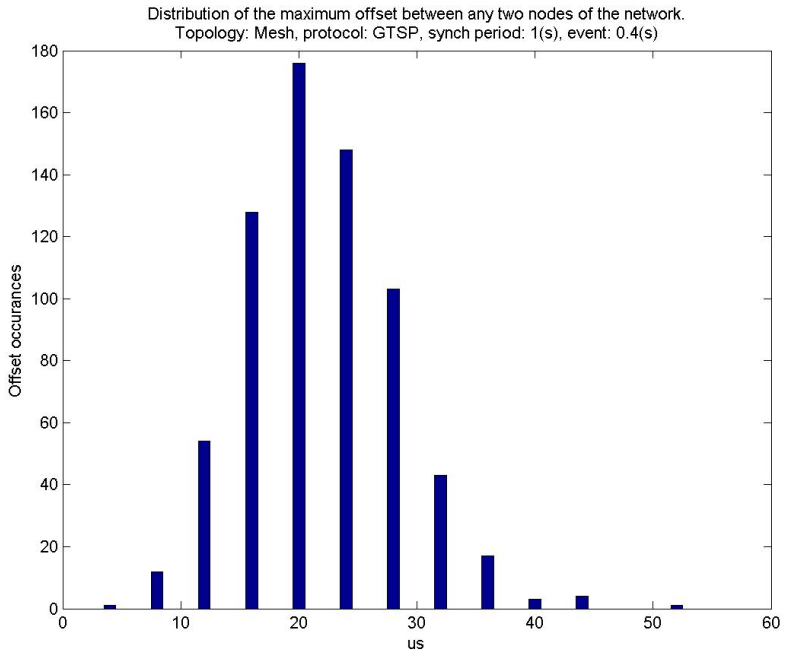

Figure 5.2: Distribution of global synchronization offsets for GTSP, linear topology and synchronization period 1 second.

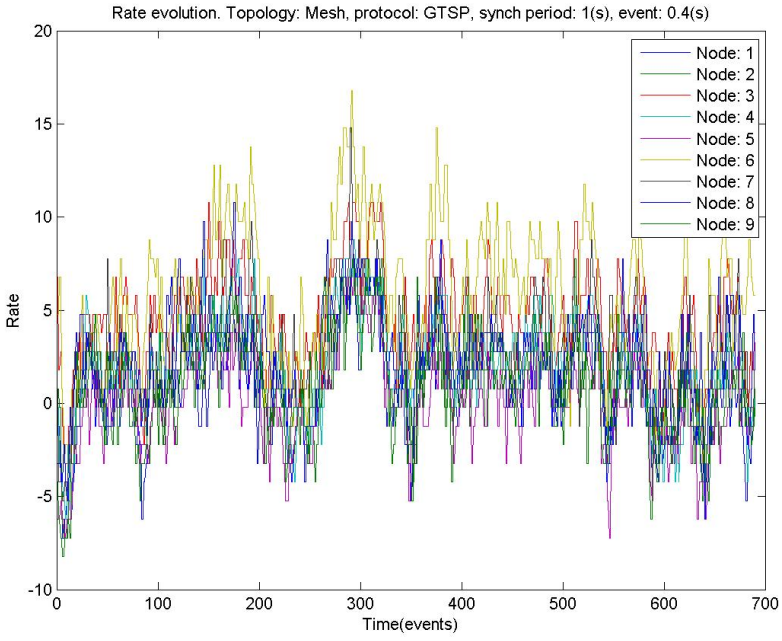

Figure 5.3: Rate evolution as the experiment progresses .

slowest nodes in the node set tested. Apart from this example, the rate evolution has been examined carefully for all tests performed, since it constitutes a core function of the protocol and improves the synchronization between the nodes in the network.

5.1.2 Sync Period and Multi-hop

Since in a multi-hop topology the nodes have less information about the global synchronization, by means of less neighbors, it is expected that the network will perform worse. The two examples are examined, i.e. example 1 and example 3, where GTSP is evaluated on tests for synchronization periods 1 and 10 seconds and for two different topologies, the mesh and linear.

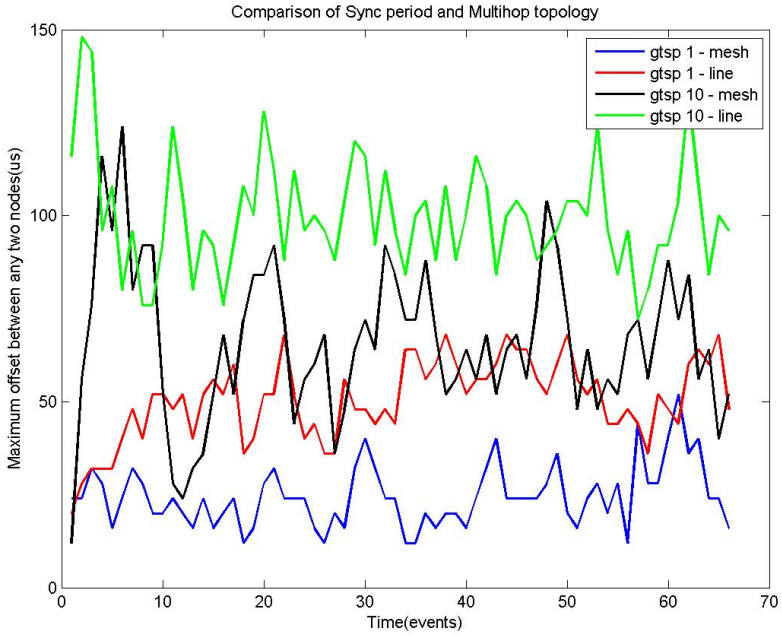

Figure 5.4: Comparison of different scenarios for GTSP based on global synchronization offset.

The Figure 5.4 depicts the maximum offset between any of the 9 nodes after all nodes have booted and exchanged their first messages. Only part of the total experiment is presented, so that the transitions to synchronization are more visible. The mean values and standard deviation is calculated for all four cases and gathered in Table 5.3.

The impact of the synchronization period in the global synchronization accuracy is important, while the network experiences bigger oscillations. For instance,

Table 5.3: Statistics for the global synchronization for different scenarios (GTSP).

Protocol GTSP	Mean (s)	Standard Deviation (s)
1 second Mesh	24.5	8.3
1 second Line	50.5	10.8
10 seconds Mesh	65.9	20.8
10 seconds Line	99.9	15.3

in the case of the 10 seconds and mesh network, most of the values are between 40 and 80 µs, a much bigger range if one compares it with the range that occupy the two scenarios with synchronization period of 1 second.

The impact of the multi-hop topology is also important. In the scenario with the 1 second synchronization period and linear topology, the system's accuracy is around two times worse than the mesh scenario. The systematic error accumulating during the hops can explain the worse performance in the cases of multi-hop topologies.

Similar experiments are done to evaluate the influence of the multi-hop topology and the synchronization period for FTSP, examining the examples 2 and 4 and the results are gathered in Figure 5.5.

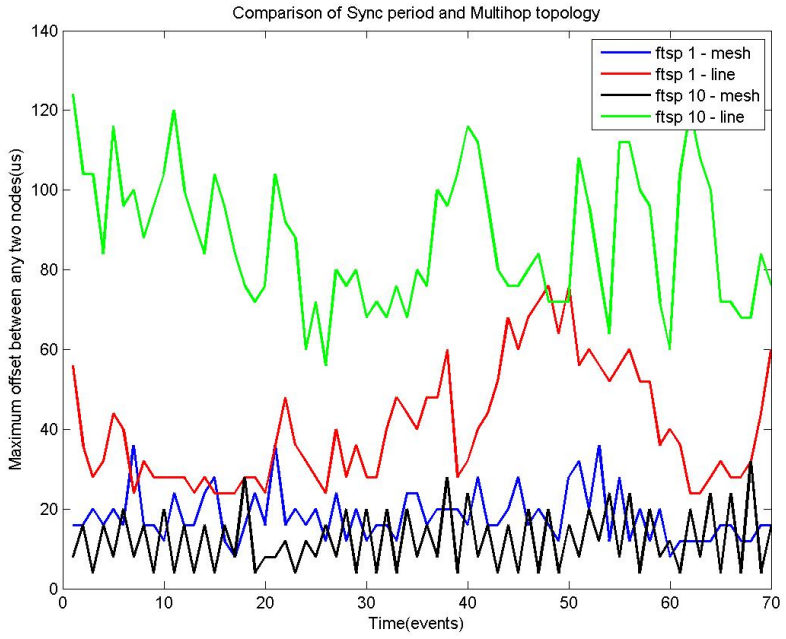

Figure 5.5: Comparison of different scenarios for FTSP based on global synchronization offset.

Better results would have been expected in the cases of the mesh network, compared to the GTSP protocol, because all 9 nodes simple follow the time of the node with the smallest id, without having to propagate or average the time with neighbors. Hence the error left is the error deriving from timestamping a simple transmission and reception, in combination with the error added by timestamping the event produced and the same error occurring in calculation of the drift. The mean values and standard deviation of the offsets are gathered in Table 5.4.

Table 5.4: Statistics for the global synchronization for different scenarios (FTSP).

Protocol FTSP	Mean (s)	Standard Deviation (s)
1 second Mesh	19.1	6.1
1 second Linear	44.9	17.0
10 seconds Mesh	12.7	7.7
10 seconds Linear	86.5	17.34

Initially, also the statistics show a much better performance in the case of the mesh network. One surprising result is the better performance of FTSP in the case of 10 seconds synchronization period compared to the case of the 1 second. For synchronization period of 1 seconds, one can see the big effect of the lost information resulting from the limited time resolution of 4µs and rounding errors. The drift compensation does not effectively help, since either the drift values calculated are smaller than the resolution available or the portion of the lost information, due to this limited resolution, is larger with respect to the drift value.

On the other hand, in the case of the 10 seconds the portion of the lost information with respect to the drift value is much smaller, allowing for better correction of the synchronization offsets. Encouraging is the fact that the drift compensation is accurate, confirming the proper implementation of this component.

5.2 Lossy Environment

The framework evaluates the performance of the two protocols also in lossy environments. Two types of losses are considered for evaluation, the random losses and burst error losses. For the result section, the configurable threshold is set to 1ms. Considering the target application of power grid line protection, this threshold translates into achieving 20 distinguishable samples per waveform of the current.

5.2.1 Scenarios

The evaluation of the protocol is performed in real time, which means that each test lasts several hours to be performed. Due to time limitations, a selection of test scenarios are performed, which is gathered in Table 5.5.

Table 5.5: Selection of tests evaluated.

Scenario	Protocol	Topology	Loss Type	Sync Period
1	GTSP	Linear	Random Packet Loss	10 seconds
2	GTSP	Linear	Random Node Isolation	10 seconds
3	GTSP	Double line	Random Packet Loss	10 seconds
4	GTSP	Double line	Random Node Isolation	10 seconds
5	FTSP	Linear	Random Packet Loss	10 seconds
6	FTSP	Linear	Random Node Isolation	10 seconds
7	FTSP	Double line	Random Packet Loss	10 seconds
8	FTSP	Double line	Random Node Isolation	10 seconds

For all scenarios tested, same configuration parameters are chosen. In Section 4.2.4 it is explained that after each time a certain lossy environment is evaluated, the nodes are rebooting. The number of synchronization periods after the reboot, time during which the nodes have to synchronize before the next lossy environment is evaluated, is chosen to be 30. This value is based on experimental results, which show that it is sufficiently large, to allow for tight synchronization.

The results from random packet losses and the node isolation duration are presented in the next sections.

5.2.2 Random Losses

In this scenario the communication between the nodes in the network experience random loss. Each node drops packet with a specific probability and the influence of these lossy conditions on the network synchronization is examined.

Impact of loss “pulse”

Initially a first comparison between GTSP and FTSP is presented, while zooming in the reaction of the network synchronization for each protocol under a specific lossy environment. More specifically, the case of the linear topology is examined, with 10 seconds synchronization period and random packet losses of 60% and 80%. Figure 5.6 includes measurements corresponding to FTSP. On the top part of the figure, the global synchronization accuracy is depicted, with and without the use of drift information when reporting the events, while the lower part shows the probability of that the network nodes are dropping a received packet.

Noticeable is that although the packet losses influence the global synchronization offset, the maximum offset in the worst case stays below 2ms for most of the cases.

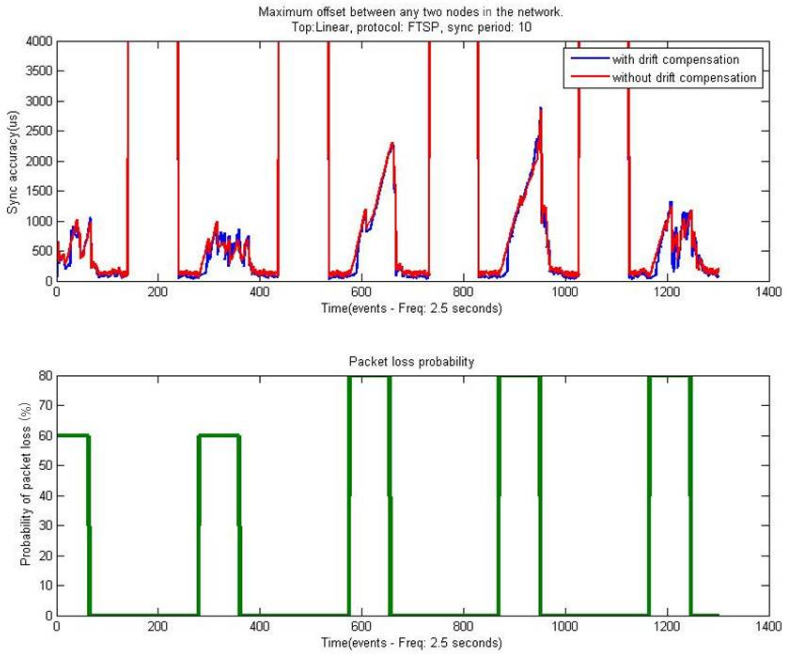

Figure 5.6: FTSP example influence of random packet losses "pulse" on the synchronization accuracy.

Moreover in Figure 5.6 can be confirmed that the network's synchronization may be influenced to a different degree, depending on which links are affected. This issue is addressed in 4.2.4.2, where the necessity to repeat the same lossy conditions several times is pointed out, in order to collect more representative results. To bring FTSP and GTSP in comparison, Figure 5.7 shows that GTSP is more influenced by losses. The nodes after few rounds under losses they desynchronize, with the global error exceeding the 4ms. Values exceeding 4ms are capped to 4ms for the sake of better readability.

During the measurements the nodes report the timestamps of events both before and after correcting them using their drift information. Observing carefully the accuracy reported with and without the drift compensation, namely the blue and red line, one can see that the influence of the rate, which is the drift compensation mechanism for GTSP, does not improve the results in the lossy environments. Moreover, since the rate will correct the time that every node will broadcast during its next transmission, it can be concluded that the impact of the drift compensation algorithm deteriorates the performance of the protocol that under lossy conditions. Though this should not be surprising, since the convergence of the logical clocks to a steady-state value depends on the whether the connectivity graph of the network is strongly connected. Since the packet

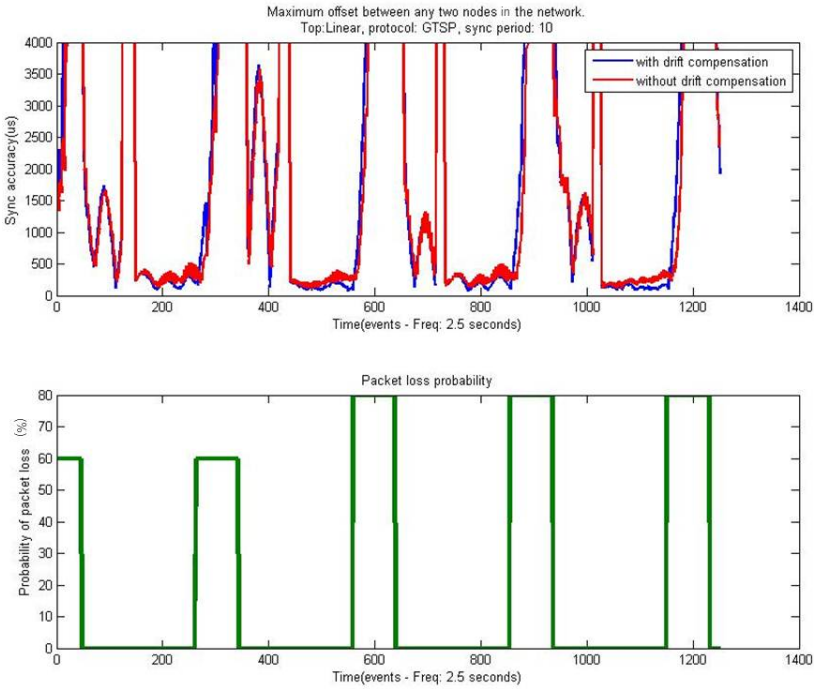

Figure 5.7: GTSP example influence of random packet losses "pulse" on the synchronization accuracy.

loss probability in the case of Figure 5.7 is high, the connectivity graph is not strongly connected, hence not being able to assure the convergence of the logical clocks. In the mitigation sections 4.3.5.1 and 4.3.5.2 it is shown how the this impact of this phenomenon can be reduced.

Impact of losses

After repeating every different each lossy environment several times, the statistics are averaged for each scenario. Collective results help more to realize how the protocols are affected by the grade of losses introduced into the network.

The framework examines the offsets that are under the threshold of 1ms and in the same time shows the percentage of the duration where the offsets were measured under the threshold. Hence a valid and fair result evaluation includes the synchronization offset metric, either this being the global, local, max or average pairwise offset with its respective percentage of time that the offsets were calculated under the threshold.

GTSP and FTSP are compared based on their global synchronization accuracy (Figure 5.8), but also on the percentage of the time instances that the system stays into the synchronization bounds of 1ms (Figure 5.9). Scenarios

1,3,5 and 7 are brought into comparison.

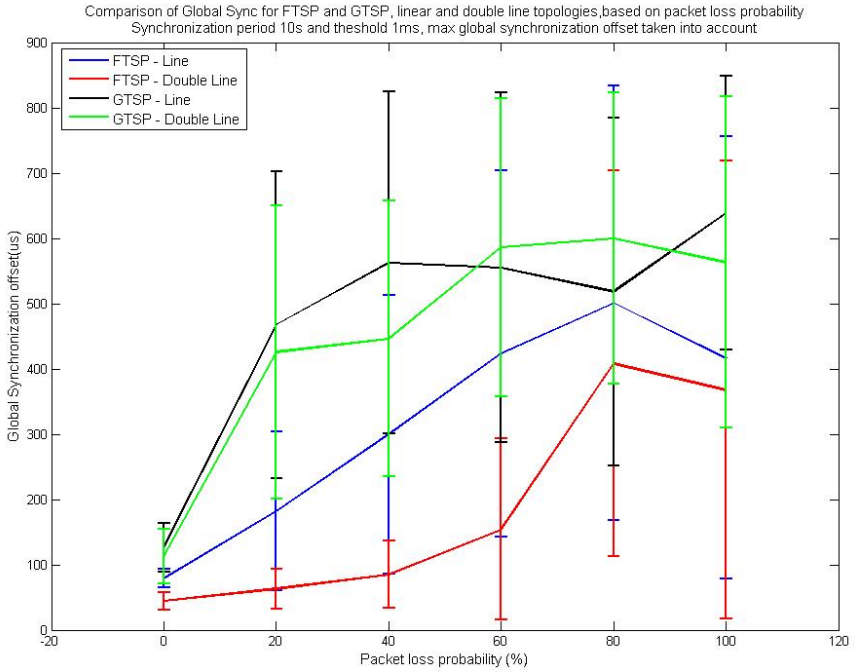

Figure 5.8: Comparison of GTSP and FTSP, linear and double line topologies, for different packet loss probabilities, based on global synchronization.

Commenting on Figure 5.8, it is observed that both topologies perform similarly in the loss free environment. However the double linear topology is in general less affected by the lossy environment. FTSP outperforms GTSP.

A closer look at FTSP shows that the error increases approximately linearly with the loss probability in the case of the linear topology, while the double line topology seems to be able to handle quite better losses up to 60%, maintaining the global synchronization error between 100 and 200μs.

On the other hand GTSP is clearly influenced by any losses in the system. Even in a lossy environment of 20%, the impact of every lost packet is bigger compared to the case of FTSP. This is a result of GTSP's concept, to average the time and rate with any neighbor.

In order to analyze this behavior, a simple example, where a node with average drift is between two nodes with clocks which run in different rates: a slow and a quick. In normal loss-free operation, the middle node does not have to make big adjustments in its time or rate and the two neighbors are trying to reach the average time. If a packet of the quick node is lost, then the impact of the slow node to the time and rate of the middle node is greater and vice

versa. The end results of such a case is that the quick node does not correct its rate anymore, averages with other neighbors and network losses the tight synchronization achieved. The impact of the drift compensation, by means of keeping track of the rate deteriorates the overall performance, not enabling to the system to converge.

This can be also noticed when the average percentage of the time, during which the network's global synchronization stays under the limitation of 1ms, is examined.

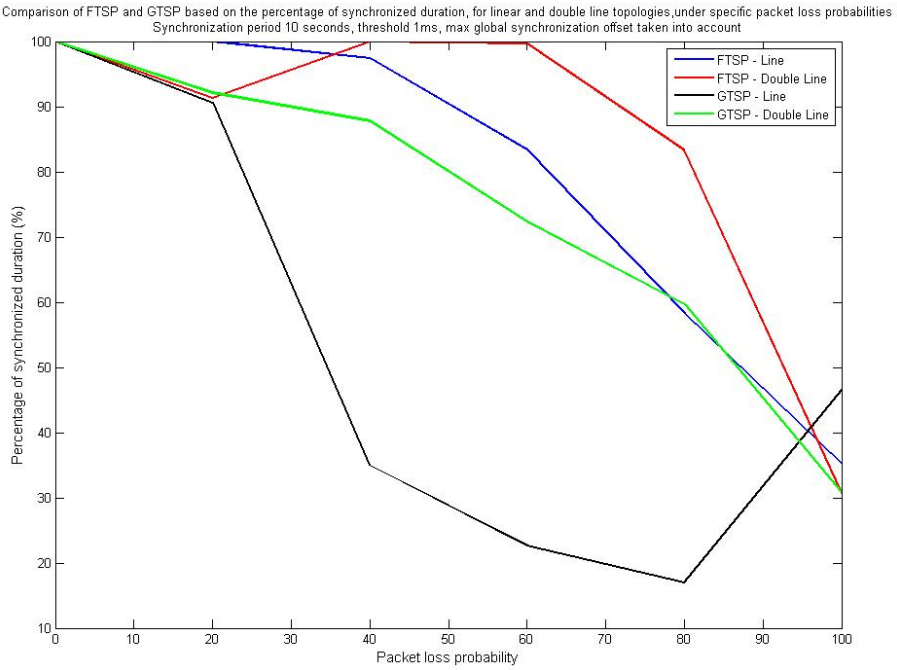

Figure 5.9: Comparison of GTSP and FTSP, linear and double line topologies, for different packet loss probabilities, based on the percentage of time duration when max global offsets are below the 1ms.

Moreover it is observed that GTSP over a double line topology is able to maintain the synchronization longer, compared to the linear topology, which shows how the double line topology can maintain the synchronization between the required limits. A conclusion that can be further made by Figure 6.9 is that GTSP desynchronizes more quickly than FTSP. Interestingly enough, it is shown that GTSP outperforms FTSP in the case of 100% loss probability for the line topology. This constitutes a special case, since no more messages are exchanged anymore. Hence the rate of each node is kept and it looks like that performing drift compensation based on this rate allows the system to maintain

synchronization longer than in cases where some packets are received randomly.

In fact, even in local synchronization, the pairwise offset of neighbors give approximately the same results. For instance the local pairwise offset is displayed in Figure 5.10 and the respective synchronization percentages in Figure 5.11.

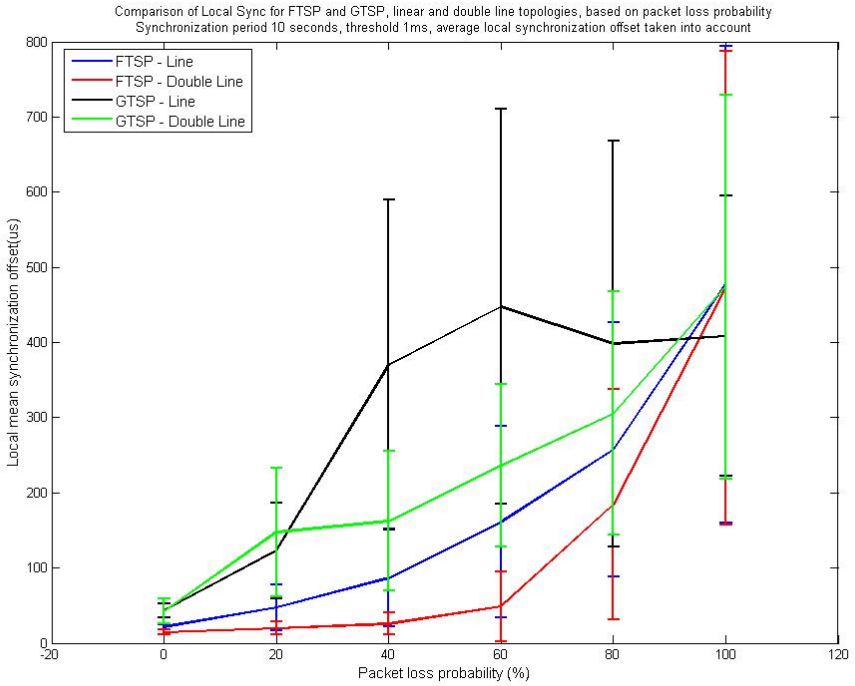

Figure 5.10: Comparison of GTSP and FTSP, linear and double line topologies, for different packet loss probabilities, based on mean local synchronization.

Evaluating based either on max global or average local synchronization offsets, the deterioration of the performance of the protocols is similar. However the values of the offsets for mean local synchronization are much smaller, fact that makes naturally sense.

Similar conclusions to those of Figure 5.9 can also be drawn based on Figure 5.11. The difference is that higher percentage of the time duration the average mean local offsets are below 1ms.

As a conclusion, on average GTSP is outperformed by FTSP. GTSP seems to be more influenced by the random packet losses. The nature of synchronization used by FTSP, where the nodes correct their clocks only when they receive a valid and updated message by their synchronized parents, helps to maintain the offsets into lower values. On the contrary, GTSP, as more distributed, performs worse since all nodes, regardless whether they are synchronized or not, broadcast their rate and absolute value of time, contributing to the time correction of their

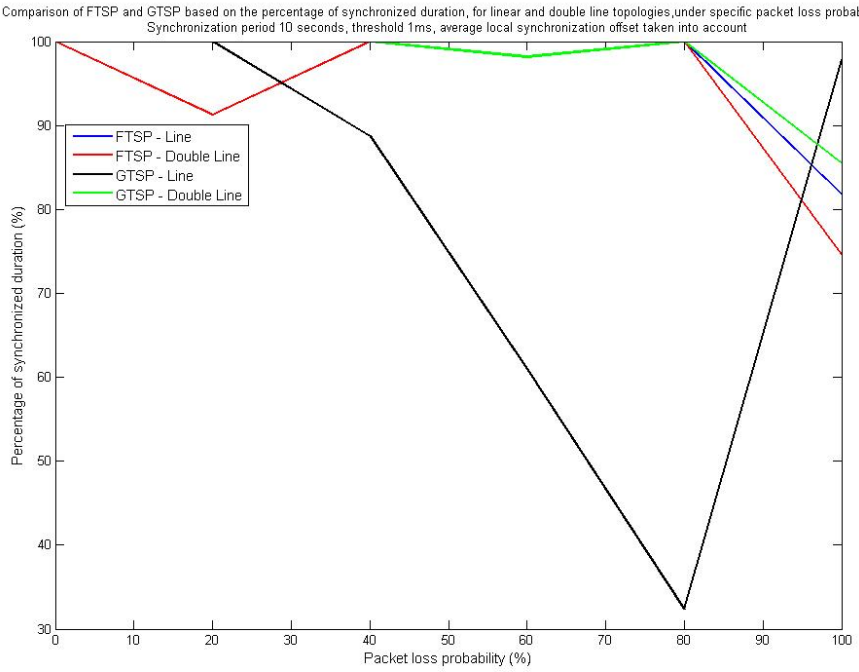

Figure 5.11: Comparison of GTSP and FTSP, linear and double line topologies, for different packet loss probabilities, based on the percentage of time duration when average local offsets are below the 1ms.

neighbors.

5.2.3 Node Isolation

For the node isolation results, where some nodes are not communicating, the framework can examine both the behavior of the whole network, including the timestamps reported from the isolated nodes, or focus on the rest network, to see how it behaves in case one of the nodes is failing. The first case provides results for scenarios where a node is obstructed or busy and still considered as synchronized, while the second case deals with scenarios where the a node is rebooting and is completely out of synchronization.

The loss configuration parameters are the same for all scenarios tested. The probability, that a node will be isolated for a certain number of synchronization periods, is chosen 20%. Since it probable that no node will get isolated during the simulated lossy environment, the results concerning the tested lossy environment take into account only measurements taken when at least one node is in isolation.

The result analysis of the node isolation is separated into two categories; the

obstructed node and the rebooting node.

Obstructed node

In this section the reported timestamps of the isolated nodes are included, assuming the failing node is able to detect an event and timestamp it. Scenarios 2, 4, 6 and 8 are brought into comparison.

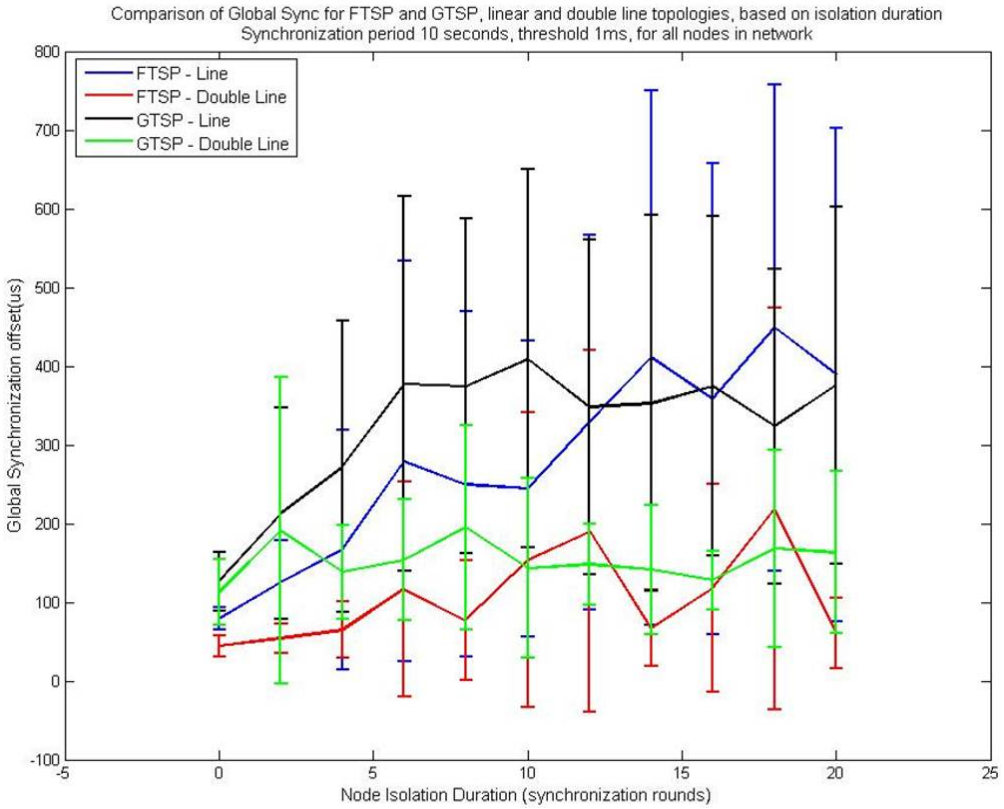

Figure 5.12: Comparison of GTSP and FTSP, linear and double line topologies, for different node isolation durations, based on global synchronization for all nodes.

In Figure 5.12, the maximum offsets of any nodes in the network are presented, for different node isolation durations. Both protocols, depicted with green and red lines respectively, perform better over a double line topology than in the case of the linear topology. In order to be able to draw safe conclusions, in Figure 5.13 the percentage of the time duration during which the max global synchronization offset remains under the limit of 1ms is presented.

Combining Figure 5.12 and Figure 5.13, it is significant to notice, that although the two protocols look like they perform similarly, GTSP is outperformed

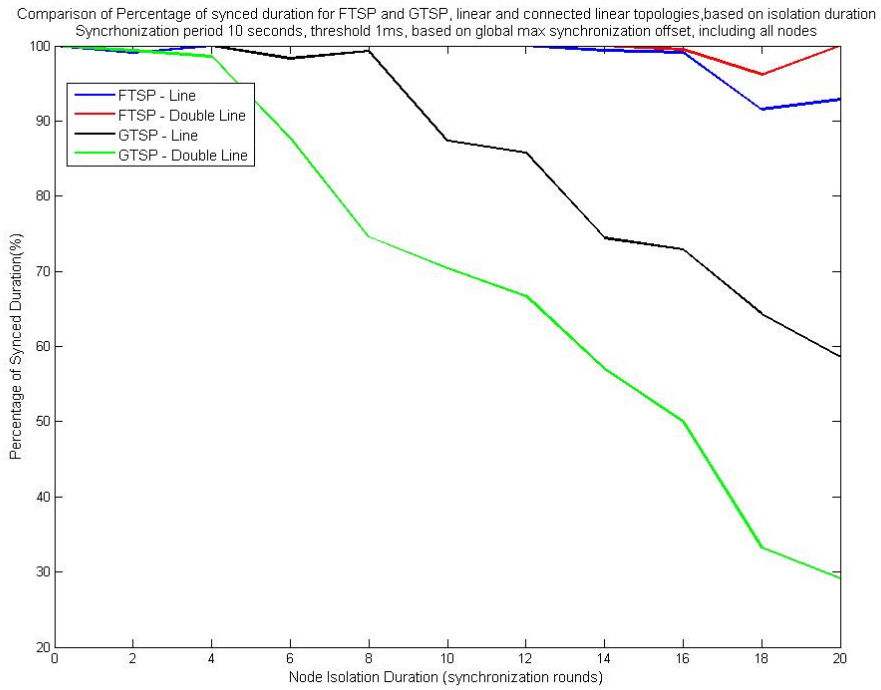

Figure 5.13: Comparison of GTSP and FTSP, linear and double line topologies, for different node isolation durations, based on the percentage of time duration when max global offsets are below the 1ms, for all nodes.

by FTSP when the percentage of the time duration, during which the offset is below 1ms is examined.

Surprising is the fact that GTSP over double line topology performs worse than in the case of linear topology in Figure 5.13. With a close look at the full test results leading to this curve, it is seen that the fact that more messages are taken into account in the double line chain before the application of the lossy condition. This results in a quicker deviation of the rate compared to its initial assigned value. The timestamps are corrected based on their rate and a common reference number, which is the initial value, and the time interval since the last synchronization point. When the losses are introduced, while the node isolation is increasing, this time interval is increasing for the nodes in isolation, resulting in larger correction for the drift compensation. Another reason for the degradation of the performance of GTSP is derived from the fact that during the absence of the obstructed node, the rest of the nodes will adapt their rates. These rates will be calculated without including the clock drift information of the obstructed node, creating a bigger gap between its time and the average time agreed by the rest nodes. Although this is observed both for the linear and double line topology,

the effect is quicker for the double line, since the connectivity is stronger and more time information is used for the rate modification.

Hence it can be concluded that FTSP in a network with the double line topology can maintain the max global synchronization offset under the limit of 1ms for almost all the time and even more specifically performing in the range of 100-200μs, an accuracy satisfying the requirements of most applications.

Rebooting node

In this section the reported timestamps of the failing nodes are not included, assuming that are completely desynchronized. Only the rest of the network is examined to see whether it can maintain its synchronization, regardless the fact that some of its nodes have stopped communicating. Scenarios 2, 4, 6 and 8 are examined, though for this example the maximum global synchronization offset, including the reported timestamps of the nodes in the network that are synchronized.

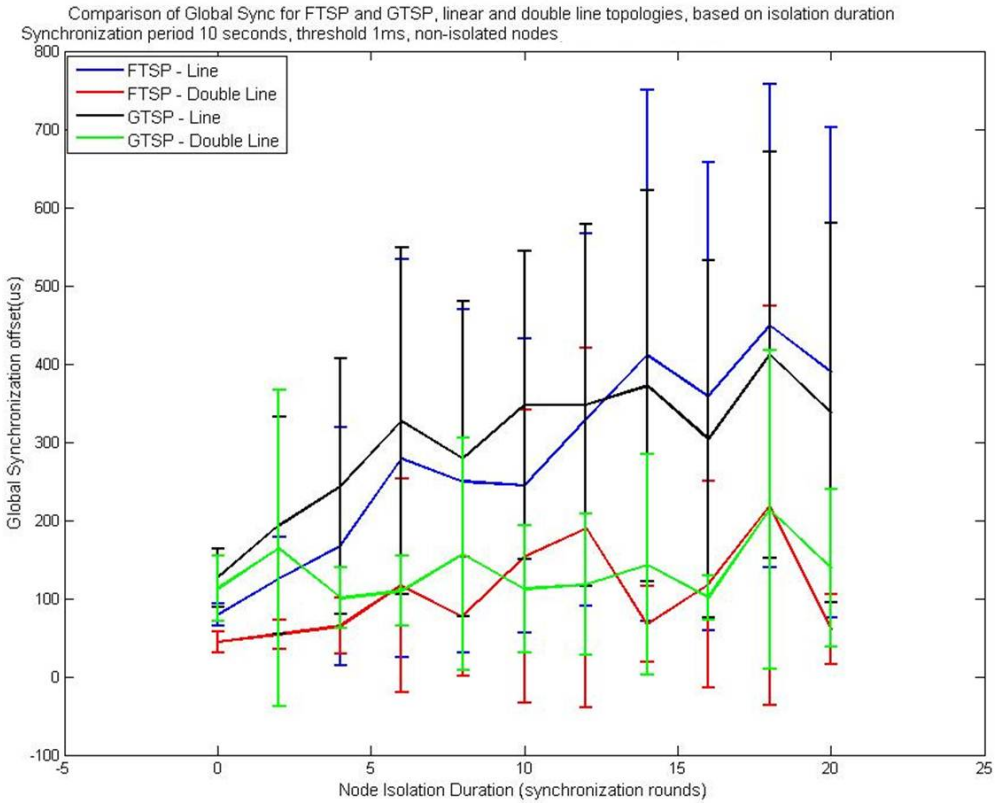

Figure 5.14: Comparison of GTSP and FTSP, linear and double line topologies, for different node isolation durations, based on global synchronization for non-isolated nodes.

Figure 5.14 depicts the maximum offset between any two nodes in the network and its influence depending on the duration, in synchronization rounds, of the nodes' isolation. It can be seen that the double chain topology can handle the isolation of some of the network's nodes, performing always around 100µs, slightly increasing with the increase of the isolation duration. On the other hand, the linear topology is more influenced, since the network is split in two during the node isolation. Each network part is synchronizing separately, leading to larger synchronization offsets.

The percentage of duration that the nodes are synchronized for the same metric and scenarios is depicted in Figure 5.15.

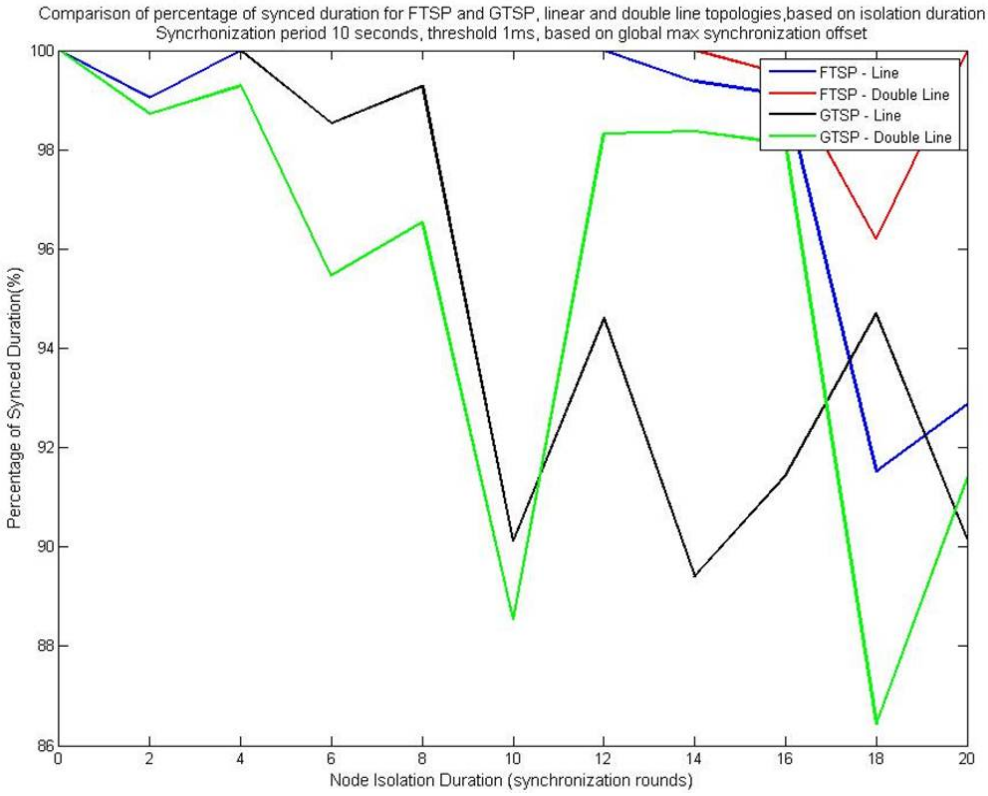

Figure 5.15: Comparison of GTSP and FTSP, linear and double line topologies, for different node isolation durations, based on the percentage of time duration when max global offsets are below the 1ms, for non-isolated nodes.

Although comparing the two protocols based on their offsets would conclude that they perform similarly, the percentage of the time duration while the global offset is below 1ms is worse for GTSP. However this percentage is higher than 85%, leading to the conclusion that for these scenarios the synchronization between the nodes that do not fail is close to meet the required limit of 1ms.

Chapter 6

Conclusion

In this thesis, the components required to build a framework that is able to evaluate the performance of synchronization protocols under lossy links are presented in detail.

Afterwards the framework is applied in a test case evaluation of FTSP and GTSP and provides results for analysis. The models of lossy links evaluated in the test case are the packet losses with a given probability and node failures for a given duration.

An analysis of the importance of the time resolution is performed and it is shown using a simple example how it can affect a protocol's synchronization performance, mainly as the network's diameter is increasing. Similarly, the necessity of minimizing the variance of the delay between a transmission and a reception of a packet is explained. This delay is measured for the tested platform and used to improve the achievable protocols' performance.

Furthermore, the clock drift, the phenomenon where the clock of a node does not run at the exact right speed compared to other nodes' clocks, is explained. A typically used methodology for a node to calculate its drift compared to another node is presented and the actual drifts of the nodes are measured.

Linear and the double line topologies are evaluated for both the FTSP and GTSP protocols based on metrics such as the maximum and average offset between any two nodes in the network. The results show that the random packet losses have a strong effect on both protocols' performance. One the other hand, failing nodes do not have as large an influence on the performance of the protocols, with FTSP used in a network, where nodes are connected in the double line topology, outperforming the rest of the cases.

Recommendations for future work include investigating a hybrid system that implements multiple synchronization protocols, as this could potentially improve the performance. Additionally, the framework could be extended to be able to evaluate real multi-hop topology. To achieve that, multiple external nodes must be connected in such a way, so that all the trigger nodes are simultaneously

informed to produce an event. Such an implementation would require an investigation on the possible ways to achieve a connection with low variance in the propagation delays, reassuring that the trigger reaches all the external nodes simultaneously. Possible solutions would be to use wireless transmissions with higher power or wired communication. The framework presented in this thesis could be a useful component of such a system, since the required modification includes only the extraction of the whitelists.

Bibliography

[1] M. Z. Zamalloa and B. Krishnamachari, "An analysis of unreliability and asymmetry in low-power wireless links," *ACM Transactions on Sensor Networks (TOSN)*, vol. 3, no. 2, p. 7, 2007.

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

[3] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, 2002.

[4] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 138–149, ACM, 2003.

[5] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, ACM, 2004.

[6] J. Sallai, B. Kusỳ, Á. Lédeczi, and P. Dutta, "On the scalability of routing integrated time synchronization," in *Wireless Sensor Networks*, pp. 115–131, Springer, 2006.

[7] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pp. 73–84, IEEE, 2011.

[8] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 142–153, ACM, 2005.

[9] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pp. 37–48, IEEE Computer Society, 2009.

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

[11] C. Lenzen, T. Locher, P. Sommer, and R. Wattenhofer, "Clock synchronization: Open problems in theory and practice," in *SOFSEM 2010: Theory and Practice of Computer Science*, pp. 61–70, Springer, 2010.

[12] O. Mirabella, M. Brischetto, and A. Raucea, "Evaluation of clock synchronization protocols for wireless sensor networks," in *Wireless Days (WD), 2009 2nd IFIP*, pp. 1–5, IEEE, 2009.

[13] L. Redwire, "Redwire." `http://www.redwirellc.com/`. Accessed: 2013-04-11.

[14] A. Dunkels, "Contiki: The open source operating system for the internet of things." `http://www.contiki-os.org/`. Accessed: 2013-04-11.

[15] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 455–462, IEEE, 2004.

[16] L. Ramadoss and J. Y. Hung, "A study on universal serial bus latency in a real-time control system," in *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pp. 67–72, IEEE, 2008.

[17] A. Dunkels, "Mobile econotag." `https://github.com/contiki-os/contiki/wiki/Mobile-Econotag`. Accessed: 2013-04-11.

[18] A. Dunkels, "Timers." `https://github.com/contiki-os/contiki/wiki/Timers#wiki-The_Etimer_Library`. Accessed: 2013-04-11.

[19] D. Dzung and G. Kalman, "Concepts for timing over lossy and low power links." Technical Report 9ADB005875-001, ABB Corporate Research, 2013-01-18.

[20] Freescale, "Mc1322x reference manual." Rev. 1.6.