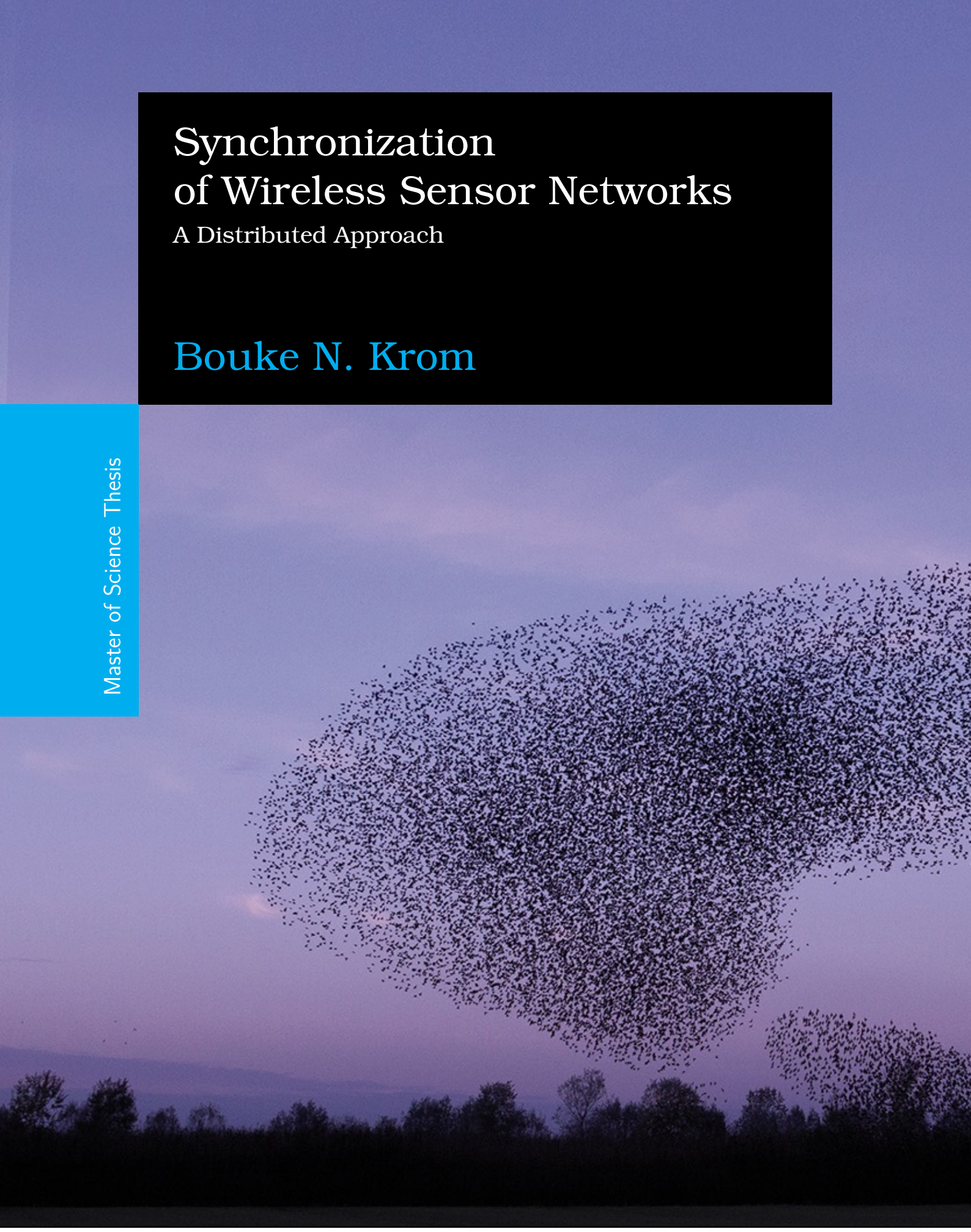
Synchronization of Wireless Sensor Networks

A Distributed Approach

Bouke N. Krom

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Synchronization of Wireless Sensor Networks

A Distributed Approach

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Bouke N. Krom

May 16, 2015

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Chess Wise B.V.

Delft University of Technology
Department of
Delft Center for Systems and Control (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Synchronization of Wireless Sensor Networks

by

Bouke N. Krom

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: May 16, 2015

Supervisor(s):

Dr.ir. Manuel Mazo

Dr.ir. Robert Jan Gorter (Chess Wise)

Dr.ir. Martin Streng (Chess Wise)

Reader(s):

Prof.dr.ir. Hans Hellendoorn

Marco A. Zuñiga Zamalloa PhD

Abstract

Wireless sensor networks of the type discussed in this MSc project play a crucial role in many envisionings of the Internet of Things, a trend that is thought to play a major role in the technological innovations of the near future. These wireless, ad-hoc, scalable mesh networks provide the infrastructure for numerous sensing and control applications. A key requirement for such networks is energy efficiency. Synchronization of nodes can significantly improve energy efficiency by enabling a tighter communication schedule.

In this MSc project an improved synchronization algorithm for an existing MAC protocol is developed. After establishing a clock model and surveying the literature for existing algorithms, the synchronization problem is modelled from a control theoretic viewpoint. It is shown that the synchronization problem closely resembles the consensus problem, which is extensively covered in literature. This insight is used to prove stability of a class of synchronization algorithms – including the existing algorithm – under requirements on the communication topology that are easily satisfied.

A set of improved algorithms is developed, and their performance is assessed in simulations. The best performers were tested experimentally on networks with up to 300 nodes.

In conclusion, we have been able to create a substantial improvement over the existing synchronization algorithm, attaining a higher throughput and longer network lifetime. Experiments have shown however, that very large networks (>150 nodes) are not adequately described by our models and can display unexpected dynamics.

Contents

Acknowledgements

This thesis marks the culmination of an important stage of my life. One that took considerably longer than expected – but not for lack of progress. I want to thank my supervisors Robert Jan, Martin and Manuel for the guidance, coaching and fruitful discussions concerning this MSc project and many other things. The process hasn't always been smooth, and I want to thank my family (both north and south) for their support.

Sometimes a small nudge can be the start of a new trajectory. For some of these nudges, I would like to thank Matthijs, Wolf, George, Siebren, João, Thijs and Karel. Your (sometimes unintended) advice has brought me where I am today. And most importantly, there is no way I could have made it without the love and patience of my girl. I hope I can mean the same to her.

Delft,
May 16, 2015

Bouke N. Krom

Chapter 1

Introduction

Over the last couple of decades, the technological advancement of our society has enabled a world that is increasingly connected. The internet – possibly the most influential recent invention – has moved from an academic novelty in the early nineties to a technology that approximately 97% of Dutch people have access to today[10]. The trend to exchange data in networks is continuing. It is expected that the Internet of Things (IoT), connecting all kinds of devices to each other and the internet, will be a defining movement in technology in the coming years [73, 6]. World-renowned firms and consortia are competing with different standards and protocols to get into these markets early [56]. This also sparks debate about the role these technologies and firms will have in our lives [50, 31]. Wireless, ad-hoc, scalable mesh networks such as the Wireless Sensor Networks mentioned in the title of this thesis are a category of networks in this broad field of 'connected things', and can provide an infrastructure for numerous sensing, control and automation applications.

1-1 Wireless Sensor Networks

A Wireless Sensor Network (WSN)[1] typically consists of small, energy efficient sensors, communicating wirelessly with each other, that can be deployed in large numbers to sense or actuate a physical phenomenon in a spatially distributed way. Typically, these networks form their own infrastructure (ad-hoc), are robust to failure of individual nodes or links, and are scalable to hundreds or thousands of nodes. This places high demands on all aspects of hardware and software design, and it wasn't until the start of this century that it became possible to construct these networks. The potential of these possibly ubiquitous networks was immediately recognized by academia [71], and a new field of research emerged.

Nowadays, applications with WSNs start to become commercially viable. Products are developed for markets as diverse as street lighting control, robotic networks, asset tracking, indoor climate monitoring and wearable sensors for health and safety tracking.

An important demand in all those applications is energy efficiency. If the nodes are battery-powered, the cost of a network is determined in part by the lifetime of these batteries. If the nodes are mains-powered, attaining a high throughput while using a

[1] A WSN can often control some actuators too, and in that regard it would be more correct to talk about a Wireless Sensor and Actuator Network (WSaN). The term WSN is more often used however, even for actuating networks.

Figure 1-1: Application areas for wireless sensor networks: track and trace and climate monitoring (promotional pictures from www.chess.nl)

negligible amount of energy is very desirable. One way to improve the energy efficiency is by improving the synchronization. A tight synchronization can make the timing of the communication protocol more accurate, requiring less slack and thus less energy spent on operating the wireless link. In the same manner, the amount of data that can be communicated in a certain timespan can be increased by having an accurate timing.

In this MSc project, an improved synchronization algorithm is developed for an existing WSN infrastructure: MyriaWise, a product of Chess Wise.

1-2 Chess Wise

Chess Wise is a firm specializing in the creation of WSN backbones for diverse applications. A multi-purpose hardware platform running specialized embedded software serves as the wireless interface for many different products. The software stack, called MyriaNed, already contains a synchronization protocol. This heuristic protocol stems from the early days of MyriaNed, and engineers at Chess Wise expect that improvement in terms of performance is possible.

MyriaNed is built on several core principles, and this limits the number of viable synchronization approaches. These core principles include a globally synchronous Time Division Multiple Access (TDMA) communication scheme, no single point of failure (any node should be able to fail without breaking the network) and lifetime predictability.

1-3 Synchronization

The study of synchrony has a very long history. One of the first written accounts is that of Christiaan Huygens, who noticed that two pendulum clocks on the same table would synchronize their swinging pendulums over time (albeit in antiphase), in 1756.

With the advent of distributed computing systems in the 1970's and 80's, methods for synchronizing events over these systems were invented. Around the same time, the mathematically elegant Kuramoto model of synchronized oscillators was published, from which synchrony could emerge in seemingly mysterious ways (just as Huygens clocks). The model has since then been intensively studied, and continues to provide new insights to this day.

The phenomenon of synchronization has also garnered renewed interest in more recent years. The synchronization methods for wired distributed systems were not suitable for wireless networks, prompting the development of numerous new synchronization algorithms. The study of complex network theory emerged contemporaneously, and finding the boundary between success or failure in attaining synchrony on these networks has been a major research topic.

1-4 Goal

The goal of this graduation is to develop a new synchronization algorithm for MyriaNed, that can more effectively synchronize nodes on a large scale, dynamic, probabilistic, switching topology network.

In such networks there is a need for specialized algorithms. Wired network synchronization protocols, like the Network Time Protocol (NTP) [41] or the Precision Time Protocol (PTP) [15] are not suited to the challenges of wireless networks. NTP is not precise enough, achieving milliseconds or tens of milliseconds accuracy, whereas an accuracy of at least one order of magnitude higher is wanted. Moreover, it is a centralized protocol relying on one reference node, which defeats the robustness inherent in the distributed nature of MyriaNed. PTP is designed for enabling sub-microsecond synchronization in local networks for measurement and control, but requires diverse clocks (which are not commonly found in WSNs) and is not robust against failure of nodes, uncertain communication channels and dynamic topologies [15]. Dedicated per-node solutions such as GPS receivers or other high-precision clocks consume too much energy. Algorithms specifically for WSNs have been developed, but they are strongly dependent on the specifics of the other network components (specifically the MAC protocol). Moreover only very few of these algorithms have been tested in a large scale, realistic experimental setting.

The environments in which WSNs are deployed are varying, and often extreme. The communication topology and wireless link reliability (which both influence the performance of a synchronization algorithm) are hard to predict and recreate in experiments or simulations. In order for improved algorithms to be included in MyriaNed, a thorough confidence concerning their performance and reliability in all possible deployments must be attained. It is therefore interesting to investigate the stability and convergence properties of algorithms from a mathematical perspective: this might lead to guarantees of a minimal performance under broad conditions that can be satisfied in practice. Since experience has shown that large wireless sensor networks sometimes behave quite differently than predicted by models, doing experiments will be a crucial part of the assessment of algorithms too.

1-5 Research Approach & Outline

In order to achieve the goal stated above, the following items have been pursued:

- A literature survey to gain insight in tools and methods for analysis of synchronization algorithms on wireless mesh networks.
- An analysis of stability, convergence, robustness and/or precision of synchronization algorithms on the class of networks that encompasses MyriaNed.
- A simulation of suitable algorithms.
- A series of experiments, including a large-scale experiment (200+ nodes), to verify claims from analysis and simulation.

This thesis starts by introducing the fundamental components of the synchronization problem in chapter 2: the characteristics of oscillators, the clock model that is adopted and some terminology surrounding synchronization and wireless networks. In chapter 3, the role of synchronization in MyriaNed is discussed, eventually leading to an estimation of the possible improvements in throughput (section 3-5) and battery lifetime (section 3-6). In the next chapter the requirements on algorithms are made explicit, and the literature on synchronization algorithms is surveyed for possible improvements. We conclude the chapter with a short feasibility study on using rudimentary temperature sensing as a feed forward add-on to the synchronization algorithms. In chapter 5, a mathematical model the synchronization problem is defined. This model is then used in an analysis of requirements for stability. Returning to the algorithms of chapter 4, chapter 6 introduces three candidate synchronization algorithms, using the notation that was introduced in chapter 5. Two of these algorithms are adaptations of published algorithms, and one is a new algorithm that builds on the foundations of the existing synchronization protocol. The temperature compensation rule is described too. These new algorithms are then tuned and compared to each other and the current situation in a simulator in chapter 7. One of the candidates drops out in this stage. The two remaining algorithms are evaluated in experiments in chapter 8, and these results are compared with the simulations. Of course the last chapter lists the conclusions of this project, along with recommendations for further inquiry.

1-6 Prior Work

Before this MSc project was started, the author did an internship at Chess Wise about the same topic. This internship will sometimes be mentioned, and has largely functioned as an orientating study into the synchronization problem and the parameters of influence.

The first part of this MSc project consisted of a literature study, for which a separate report was delivered. This report is not publicly accessible. In the interest of readability, we have not referenced that study in this report, but instead reproduced the results where applicable. Especially in the chapter on algorithms (4) large parts are copied from the literature study.

The synchronization of MyriaNed has been investigated before. Synchronization with a larger granularity (the joining of networks) is a major subject in [14], which is a prerequisite for all the work done in this project. The more fine-grained synchronization was studied before in another MSc project [4], but unfortunately the very limited documentation made it impossible to build on these results. In addition, the main conclusion of that work received serious criticism by peers [29]. Furthermore, a study has been done on model-checking the synchronization in MyriaNed [30]. The conclusion of that study is that the

synchronization can be instable in a very specific three-node network, but this has little relevance for real deployments. Larger networks were too computationally intensive to be analyzed with these methods.

To our knowledge, this is the first study applied to MyriaNed in which (1) a literature survey for synchronization algorithms is done, (2) a mathematical model with a control theoretic perspective is developed and (3) large scale experiments are done with new synchronization algorithms.

Chapter 2

Fundamentals of Synchronization

Synchronizing is defined as follows by the Merriam-Webster dictionary:

syn·chro·nize

: to cause (things) to agree in time or to make (things) happen at the same time and speed

: to happen at the same time and speed

In a wireless network context, this will necessarily mean that time needs to be measured by clocks. Moreover, the above definition leaves room for interpretation and refinement. In this chapter we will introduce a clock model and explore the scope of the initial problem statement by defining some of the assumptions and principles surrounding synchronization.

2-1 Oscillators

Electronics keep track of time by using oscillators. The number of cycles of a regularly oscillating device is counted, leading to an estimation of time elapsed. A very common oscillator is a quartz oscillator, since they have a nice balance between price and performance. MyriaNodes are equipped with a 32 kHz quartz crystal that can run on very little energy when the node is idle, and that can wake up the processor (which also involves turning on a 16 MHz oscillator) when a period of activity should be started.

It is the accuracy of this 32 kHz quartz oscillator that is crucial for the synchronization in MyriaNed, as will become clear in chapter 3. The actual momentary frequency of an individual quartz oscillator differs from the nominal frequency due to several effects [70]:

- **Calibration:** due to manufacturing variability, the frequencies of crystals will not be exactly the nominal frequency.
- **Aging:** since a crystal oscillator is essentially a mechanical device, it will wear over time. Stress relief and small contaminations of the materials lead to a small change in frequency over the years of use.
- **Temperature:** the temperature of the crystal has a large influence on its frequency. There is a turnover temperature at which the oscillator has its nominal frequency, and the crystal oscillates slower as it is further removed from this temperature.

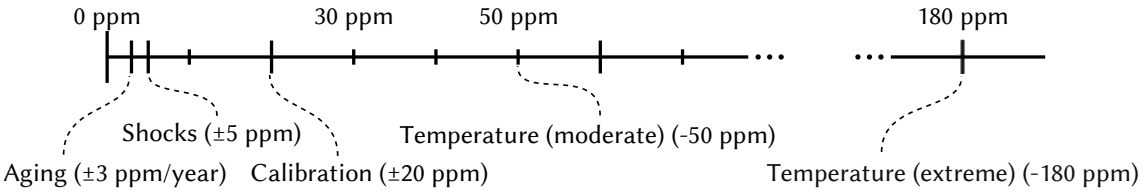


Figure 2-1: The magnitude of different influences on oscillator frequency. Moderate temperatures are at $-15\,^{\circ}\mathrm{C}$ or $60\,^{\circ}\mathrm{C}$. Extreme is at the outer operating range; $-25\,^{\circ}\mathrm{C}$. (sources: [58, 25])

- **Other effects:** there are several other influences on quartz oscillators that create variability, but these effects are usually not persistent, and thus the effect will not accumulate over time. These influences include mechanical vibrations, radiation, magnetic fields and noise processes in surrounding circuitry.

The typical magnitude of common effects is shown in Figure 2-1. Deviations are commonly expressed in parts per million (ppm).

Quartz oscillators are available in many varieties. Different calibration accuracies can be ordered, but there are also types with a temperature compensation circuit (TCXO) or even oven-controlled (OCXO) ones. Since cost per unit is important, the variety commonly used in MyriaNodes has a reasonable calibration, but no integrated compensation mechanisms.

2-2 Clock Model

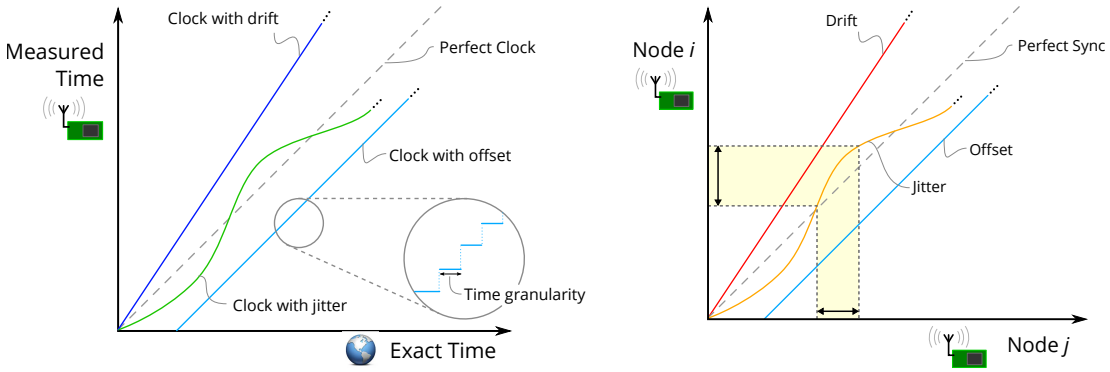
The largest disturbing effects on oscillators vary only slowly over time, especially when compared to the frequencies involved. It is therefore very common to model an oscillator-driven clock linearly:

$$\tau(t) = at + b$$

Here τ is the value of the clock, t is the real time and a and b are the drift (frequency deviation) and offset (startup difference) respectively. This affine clock model will be extensively used in chapter 5. It is good to keep in mind that although a is considered a static parameter over short time spans, it will vary considerably over the lifetime and environmental temperature of a node.

These drift and offset effects with respect to a perfect external time are displayed in Figure 2-2a. The (varying) drift and offset, together with possible noise effects are sometimes called jitter. Because the oscillator has a frequency of 32 768 Hz, one tick is 30.5 µs, the time granularity. From the point of view of the nodes, the difference between a tick that just started and a tick that is almost finished is indistinguishable. This quantization effect is further discussed in subsection 5-1-4. In Figure 2-2b a similar diagram is shown for the clocks of two nodes in the network. As will become apparent in chapter 3, it is especially important that two nodes are able to count the same time duration. By making the clocks as identical as possible, this can be achieved.

During our literature research, we found only one publication that uses a different clock model. In [20], τ is denoted as the integral of previous values of drift a until time t, where a itself is varying over time according to the exponential of an Ornstein-Uhlenbeck process. An Ornstein-Uhlenbeck process is related to a Brownian motion random process. Although this model manages to capture the inherent variability of a, its physical foundations remain unclear. For our research, this model would introduce complications without providing a clear benefit, so we will not use it.



(a) Different deviations a clock can have with respect to 'real time'.

(b) Deviations that might occur between clocks. The quantity of interest here is the difference in perceived time length.

Figure 2-2: Time deviations of clocks

2-3 Terminology

The topic of 'Synchronization in Wireless Sensor Networks' is a very broad one, and both the meaning of synchronization as well as the network specifics can vary greatly. This section will give an overview of the variability within synchronization and within wireless networks, and we will define the limiting choices made for this thesis work. Most of this section is also described in the surveys by Faizulkhakov [17], Rhee et al. [54] and Karl and Willig [33].

Synchronization Concepts

Internal or External Internal time is a network-specific notion of time. All nodes in a network can be synchronized, but this internal time might not be exactly synchronized to an external time standard. As long as there is no access to this external time standard – which is usually the case with MyriaNed – synchronizing to this time is impossible. Moreover, it is not needed for most network functions: an approximate correspondence with external time is sufficient.

Permanent or By Request In some applications, a synchronization *ex post* suffices. Several nodes may observe a sudden phenomenon for example, and only after this data is gathered the timestamps need to be adjusted to a common time scale. This can be done with a synchronization-by-request scheme. In most applications, including the TDMA-timing of MyriaNed, *a priori* or permanent synchronization needs to be kept. Specific events (like the transmission of a message) need to happen simultaneously.

Partial or Total Some algorithms keep synchronization only in different regions (usually broadcast neighbourhoods) of the network, which could then be translated into one another. This might lead to problems when a network has a dynamic topology, and MyriaNed thus demands a network-wide common time scale.

Interval or Absolute A common way to define synchronization is by defining a certain correction of the hardware clock value, and attempting to have all these software clocks display the same value at the same time instant. This is an absolute synchronization. In

MyriaNed however, the most important entity is the duration of the idle interval. The absolute clock value is not important, as long as a specific duration of time is counted equally throughout the network. The round number plus the time interval since the start of the round can be used to retrieve an absolute clock value if needed. A similar distinction is made by Werner-Allen et al. [72], coining it synchronization (absolute time agreement) and synchronicity (agreement on synchronized action). It is noted that these notions are complementary: synchronized nodes can achieve synchronicity, and synchronicity can be used to attain synchronization.

Correction or Translation Synchronization usually translates into the task of correcting a local clock to approximate a common network time. Some proposals (most notably RBS [16], section 4-2) circumvent this issue by construction a translation table, where a node keeps information about its own clock with regard to its neighbours. When communication or comparison with one of the neighbours is needed, the node can move to the time scale of its neighbour by using the translation table. Taking into account the possibly very dynamic networks of MyriaNed, and the fact that one broadcast needs to address many different neighbours, such a translation scheme is not suitable for our purposes.

Network Properties

The second factor that distinguishes synchronization schemes in wireless networks is the network itself. Even within the class of wireless networks, several choices can be made that lead to different solutions.

Unicast, multicast or broadcast When all communication in a network is done in a point-to-point fashion, with a single sender and a single receiver, this is a unicast network. Algorithms for wired networks usually operate in this fashion. In a multicast network multiple, specified receivers are present. MyriaNed is strictly a broadcast network, where a single sender transmits messages to an unknown number of anonymous nodes. The receivers can know the identity of the sending node however.

Distributed or Centralized Every algorithm in a WSN is in some sense distributed, since it will have to be executed by nodes. We can make a distinction between fully distributed algorithms, where every node operates independently with local information and no special roles are present, and more centralized algorithms, where for example a reference node is elected. In the MyriaNed philosophy, there is a strong preference for fully distributed algorithms. The demands on scalability and robustness, and the diversity in deployments leave little room for successful centralized algorithms.

Symmetric or Asymmetric A network is said to be symmetric when a connection from node A to node B implies that communication the other way around is possible as well. Asymmetry can be caused by different broadcast power of nodes for example. This is not customary in MyriaNed, but due to the relatively high collision probability, strict symmetry can by no means be guaranteed.

Determinacy of delays The sending of a message is not instantaneous, and introduces a certain delay in time measurement. In some publications this delay is assumed to be a random variable, or unknown altogether. For CSMA-based MAC protocols this is a required assumption, but for gMAC it is too strong. The fixed message size and efficient handling of MyriaNed make this delay predictable.

Explicit or Implicit For some synchronization schemes, a dedicated exchange of messages is needed. A synchronization period is defined using the desired accuracy, and synchronization procedures are periodically triggered. This might hamper the functionality of an application however, and the architecture of MyriaNed demands an implicit synchronization scheme. Messages that are used primarily for applications need to be used for synchronization purposes as well.

Chapter 3

Synchronization in MyriaCore

The software of the wireless sensor networks produced by Chess Wise can be viewed as consisting of two parts: a set application-specific functions and a set of basic functions that maintain the network infrastructure. The software responsible for the infrastructure is called MyriaCore, and this is also where synchronization takes place. In addition to the network connection, MyriaCore provides services for routing and sharing data (e.g. [24]), and some interfaces with hardware and gateway equipment.

In this chapter the parts of MyriaCore that are relevant to synchronization will be introduced. The existing synchronization algorithm is described first. The various degrees in which synchronization can fail are listed in section 3-4. In order to explore the improvements that a better synchronization algorithm might yield, a model of the probabilistic features of MyriaCore is made in section 3-5, and the gains in efficiency are sketched. Finally in section 3-6 an estimation of the possible lifetime increase is done.

3-1 The MAC layer: gMAC

The architecture of a network protocol is commonly described with the OSI model [76], using layers. Not all layers from the OSI model are relevant to MyriaCore, but the lowest levels are. The actual radio communication in the physical layer is taken care of by the radio platform. The synchronization of nodes takes place in the Medium Access Control (MAC) layer, which is provided by MyriaCore's gossip MAC (gMAC) protocol. This protocol forms the basis of MyriaCore, and is described most recently in [14]. It is documented in [4, 24, 13] too, and will shortly be reproduced here.

gMAC is a Time Division Multiple Access (TDMA) protocol that attempts to coordinate the communication between nodes. Two considerations are of importance: collision avoidance and duty cycling.

Since all nodes use the same medium, only one node within 'hearing distance' should broadcast at the same time. If a node receives the messages of multiple nodes at the same time – called a collision – it cannot decipher the packets and no information is transmitted[1]. This has to be avoided to maintain a network throughput. In addition, nodes cannot receive when they are transmitting, so both uses of the radio should be balanced.

[1] The capture effect [37], where the slightly earlier message is heard if it is powerful enough, is ignored here

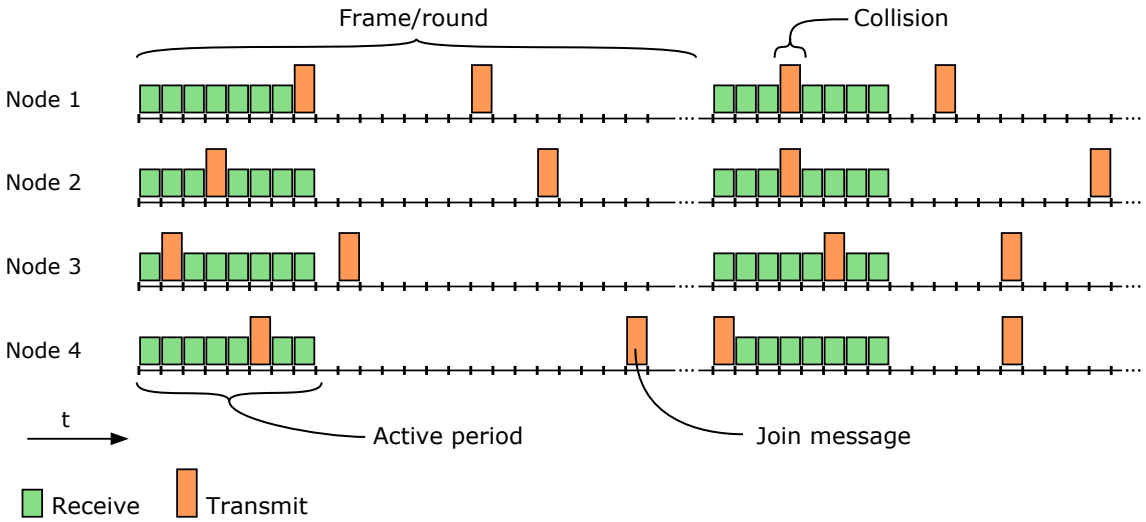


Figure 3-1: An example of the scheduling of gMAC, with some terminology. The amount of inactive slots is usually much higher than displayed, up to duty cycles of 0.5%.

In order to conserve energy, the nodes do not communicate continuously. Short periods of communication are alternated with long periods without radio activity. In gMAC each node periodically wakes up to communicate, and enters a period of sleep afterwards where no radio signals can be received. This sequence is then repeated.

Figure 3-1 gives a visual overview of the protocol. Each node executes a sequence consisting of periodic rounds or frames, typically between 0.5 s to 20 s long. Such a frame is subdivided into slots: short periods of about 1 ms. The majority of slots is spent in a sleeping or idle state, without radio communication. At the start of each frame there is an active period, where nodes power up their communication equipment. In the slots of that active period a node either listens (RX; receiving mode) or broadcasts (TX; transmission mode). This scheme can only be successful if each node has the same active period as the other nodes in the network, and if collisions of TX slots are avoided.

Per node there is only one TX slot per round, and this slot is chosen at random, effectively implementing a variation on the Distributed Slotted ALOHA protocol [55, 2]. Collisions are not deterministically avoided (such as in CSMA protocols) but only made statistically unlikely. This method is robust in the sense that given enough cycles, successful communication will eventually take place.

In areas where there are many nodes within each other's communication range however, collisions will be frequent and communication may even break down completely. To diminish the likelihood of this event, slot allocation strategies can be employed. A strategy called DistributedSlot [3] for example uses information about the number of perceived neighbours to adapt the number of active slots. Here also lies a possibility for tuning the network: if low densities are expected, less active slots will suffice and lengthen the lifetime of the network (before batteries need to be renewed). If high performance in high density topologies is demanded, a higher number of active slots is paramount, at the cost of increased energy usage.

The timing of a protocol like this needs to be precise. Even when the nodes agree on a time schedule, there will be individual differences due to the clock jitter (see chapter 2). These small time differences might lead to collisions between messages in different slots, see Figure 3-2, limiting the ratio of successful message transmissions even further. To cope with these time differences, slots are made slightly larger than the time it takes to transmit a message, see Figure 3-3. The transmission starts a small time after the start of a slot to allow nodes that are lagging behind to still receive the messages, and ends early to prevent

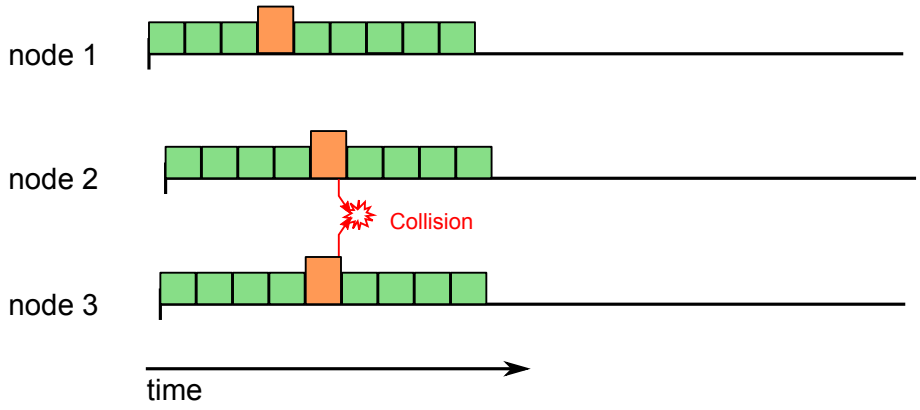


Figure 3-2: The same situation as in Figure 3-3, but without guard times. This causes collissions between messages sent in different slots.

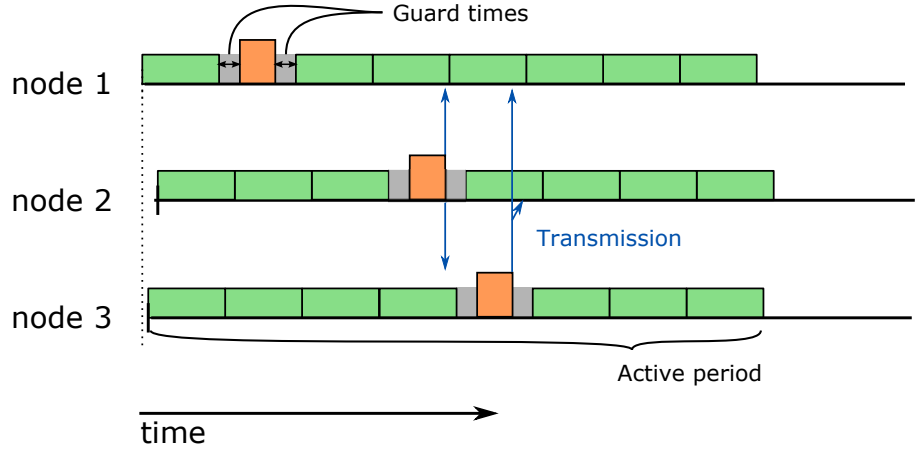


Figure 3-3: The function of guard times in gMAC: even when nodes are slightly out of sync, consecutive messages can be transmitted successfully. The case without guard times is shown in Figure 3-2

fast nodes from moving to the next slot too early. A transmission slot thus consists of a transmission time with what is called a guard time before and after it, together equalling the length of a receiving slot. If nodes are more accurately synchronized, the guard times can be made smaller. This decreases the energy spent per data byte transmitted, and thus can be used to increase the maximum data rate or decrease the energy use.

An important part of the gMAC protocol is aimed at making new nodes attain the same rhythm as an existing network, and achieving synchronization between two networks with a phase difference between their frames that is so large that the active periods do not overlap. This includes a special startup procedure to achieve initial synchronization, and the sending of a 'join' message (mentioned in Figure 3-1) during the inactive period. The scope of this project is limited to small phase differences, and the joining of nodes and networks will not be part of the research. The join procedure as it is works well, and thus we can assume for our work that nodes are already synchronized to within the bounds of the active period at the initial time.

3-2 Median Algorithm

Currently the synchronization of slots is done using an algorithm called Median. This algorithm, described by Assegei [4] and Heidarian [30], is used for keeping nodes that are

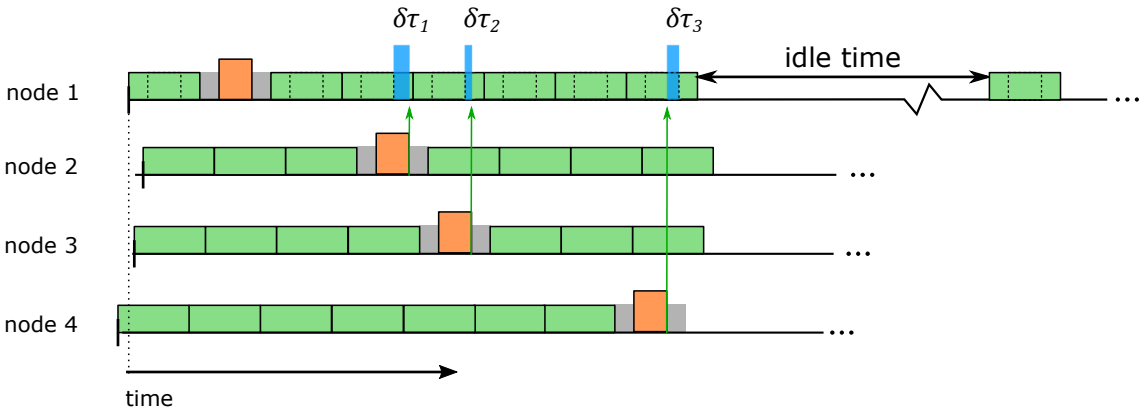


Figure 3-4: A node that receives three messages infers the time differences with those neighbours: $\delta\tau_1, \delta\tau_2$ and $\delta\tau_3$. These time differences are used to change the length of the idle time, in the Median algorithm by k_p Med$(\delta\tau_i)$, with $k_p = 0.5$.

able to communicate with each other (by having an overlapping active period) synchronized. This algorithm corrects small time differences, and we seek to improve this. The algorithm is a legacy from the more experimental days of MyriaNed, and has proven itself robust, but the performance is lacking.

Nodes compare the time of receiving a message from another node with the time at which this message was expected. This difference can be used to infer the time difference of the clocks (for a detailed description about this inference and the errors introduced see subsection 5-1-4). In each frame, a node corrects its own phase difference by taking k_p (set to 0.5) times the median of the differences with all neighbours that it received. This correction is done once: in the next frame, a new correction for the standard idle time is used, based on the newly received messages.

This process is visualised in Figure 3-4. In each slot, the node has an expected arrival time of the messages, based on the guard time and message size. The actual arrival times are recorded, and this leads to a set of time differences (one for each node received) per round. The messages that are sent between nodes always contain their slot number. This allows nodes to synchronize with Median even if messages are received in a different slot, since the correct time difference can still be recovered.

If all time differences are positive (messages arrived later than expected) this node is a little early, and should lengthen its idle time. If all time differences are negative (messages arrived too soon), the node lags behind and should shorten its idle time. Usually, the time differences will be mixed. To adapt to the majority of neighbours, the median time difference is chosen to adjust the idle time with. Moreover, initial experience showed that this prevents single outliers from skewing the rest of the network, as would happen if nodes would use the average.

The adjustment based on this median value can be tuned by the gain $k_p \in [0, 1)$. This gain is needed to prevent oscillations between nodes. If $k_p = 1$, two nodes could correct their idle time by the same amount of ticks in opposite directions indefinitely. To prevent such an exchange from happening, both should limit their adjustment by an appropriate k_p. In MyriaCore, k_p has been set to 0.5 heuristically, and this has proven to work well in practice.

Table 3-1: Guard ticks needed for certain frame times under a frequency difference of 100 ppm.

T[s]	T_g[ticks]
0.1	$\lceil 0.33 \rceil = 1$
0.5	$\lceil 1.64 \rceil = 2$
1	$\lceil 3.28 \rceil = 4$
2	$\lceil 6.55 \rceil = 7$
5	$\lceil 16.4 \rceil = 17$
10	$\lceil 32.8 \rceil = 33$

3-3 Choice of Guard Times

Controlling the energy consumption of the network starts with choosing the right guard times T_g. This is a difficult choice however. If Median is used, assuming that two nodes achieve perfect synchronisation in a frame, their perceived time difference at the start of the next frame will be determined by the combined clock drift. The accumulated time difference, expressed in clock ticks, is the drift factor, times the amount of ticks in a round:

$$\delta\tau[\text{ticks}] = \Delta f[\text{ppm}] \cdot f_n[\text{Hz}] \cdot T[\text{s}]$$

Here f_n is the nominal frequency, T is the round time and Δf is the expected drift. If we want to experience no lost messages because of synchronization errors, the amount of ticks lost in a frame must be less than the guard time. We ignore the effect of messages being sent in different slots, which might make the exact time of drifting a little more or less than the frame length (but the difference will be nowhere near the time granularity). The maximum frequency difference between two nodes caused by calibration (40 ppm), ageing (6 ppm) and reasonable temperature (50 ppm) adds up to about 100 ppm. In Table 3-1 the guard times needed under these circumstances for different frame times are listed. It seems that 9 ticks guard time (the default) is an overly conservative parameter setting for frame times below 2 seconds, and far too little for longer frame times.

It should be noted that although the drift is chosen as an extreme, this calculation assumes perfect synchronization in the previous round, which is optimistic. Moreover, it can happen that nodes fail to communicate in one round, allowing the drift to build up over two rounds. Then again, compensating all possible drift buildups between nodes with guard times will generally be too costly, and incidental errors will have to be allowed. Another factor that should be considered in this decision is the expected temperature range in which the network operates.

3-4 Desynchronization

As shown above (section 3-3), even with a synchronization algorithm synchronization errors will be a fact of life. Various degrees of synchronization errors affect the network in different ways. From small to large:

1. **Within-slot differences:** A typical slot consists of tens of clock ticks. As long as these differences are within the tolerance of the guard time, no serious consequences occur. This is the tolerable, and partly inevitable range of errors.

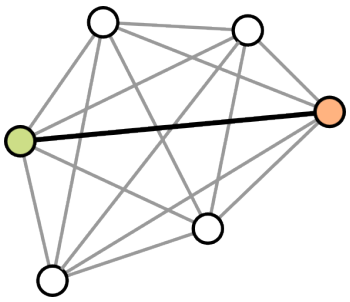


Figure 3-5: A small, fully connected network: the simplified situation in section 3-5. A transmission from the green node to the red node is considered.

2. **Slot errors:** A larger synchronization may cause messages to traverse the slot boundary: a node broadcasts a message in slot 2 for example, but several neighbours are already in slot 3 at the time of arrival. This has little consequence for the basic functioning of the network, but the data throughput will be smaller because collisions are more likely: one messages occupies two slots in which other transmissions will lead to failure. Moreover, the total window of communication is smaller because the active periods overlap only partially.

3. **Frame desynchronization:** The most serious form of desynchronization is when active periods of nodes do not overlap anymore. This can happen when nodes function a relatively long time without correction of their clocks (because no communication was possible), or when two networks that were started independently first meet. In this situation the nodes are unable to communicate, and the network is broken. Normal synchronization procedures can not cope with this, and we'll have to rely on secondary means of network unification (the join procedure).

The scope of this thesis is limited to counteracting the effects of within-slot differences and slot error (situation 1 and 2). Note that although there is a conceptual difference between within-slot differences and slot shift, a synchronization algorithm can therefore treat both cases equally.

As already apparent from the descriptions of failure modes above, the final objective of synchronization is allowing the maximum amount of data to be transmitted robustly through the network, requiring as little energy as possible. So an improved synchronization has two objectives: saving energy, thus lengthening the network lifetime, and working more efficiently. This optimization of data and energy efficiency is commonly called throughput, and in the next section we will model this tradeoff for MyriaCore. The lifetime increase will be the topic of section 3-6.

3-5 Throughput

Throughput is a performance measure that indicates how much energy is expended per message transmission: an efficiency. Since the number of messages sent and the energy expended are both constant settings of gMAC, the throughput is closely related to the probability of successful message transmission. Links in MyriaNed are highly unreliable: the probability of a sent message being received by a neighbour is far from 1. A large part of the uncertainty is caused by the mechanics of gMAC. To see where a synchronization algorithm might be of use in optimizing the success ratio, gMAC will be analyzed probabilistically in this section.

3-5-1 Probability of Successful Transmission

We simplify a deployed network (or a broadcast neighbourhood) to a group of N nodes that are all connected, as displayed in Figure 3-5. The effect of unreliable radio channels will be neglected for now, focussing solely on the effects of gMAC.

Each frame there are N_s active slots, and the number of neighbours for each node is $m_i = N - 1$. We focus on the transmission of a message between two individual nodes. Each node i has a probability of broadcasting in a slot k

$$\Pr(T_i = k) = \frac{1}{N_s}$$

With T_i a random variable denoting the transmission slot chosen.

Single link

If the network consists of two nodes, the probability of having a successful transmission is 1 minus the probability of a failure. This failure happens when two nodes broadcast in the same slot. The odds of two nodes choosing an identical slot k for broadcasting are:

$$\Pr(T_i = k \cap T_j = k) = \Pr(T_i = k) \cdot \Pr(T_j = k) = \frac{1}{N_s^2}$$

since the choice of slot is independent. However, this can happen in all N_s of the slots:

$$\Pr(T_i = T_j = k \mid k = 1..N_s) = N_s \cdot \Pr(T_i = k) \cdot \Pr(T_j = k) = \frac{N_s}{N_s^2} = \frac{1}{N_s}$$

In other words, there are N_s^2 possible combinations of broadcast choices, of which N_s lead to a collision. The probability of successful transmission from node 1 to node 2 without considering any other nodes is:

$$\Pr(T_i \neq T_j) = 1 - \Pr(T_i = T_j) = 1 - \frac{1}{N_s}$$

Fully connected network

When there are more nodes in the neighbourhood, the odds of transmission become lower. Let C symbolize the event of a collision. If each of the m_i nodes around node i picks a transmission slot at random, the chance that each of those node picks a transmission slot other than the one picked by i is:

$$\begin{aligned}\Pr(\neg C) =& \Pr(T_1 \neq T_i \cap T_2 \neq T_i \cap ... \cap T_{N-1} \neq T_i) \\ =& \Pr(T_1 \neq T_i) \cdot \Pr(T_2 \neq T_i) \cdot ... \cdot \Pr(T_{N-1} \neq T_i) \\ =& (1 - \frac{1}{N_s})^{N-1} = (1 - \frac{1}{N_s})^{m_i}\end{aligned} \tag{3-1}$$

In Figure 3-6a and b the probability described by equation 3-1 is plotted for various values of N_s and m_i. Note that the lines between points are purely there as a visual aid, in reality there are no values between the discrete points. The conclusions are as was expected: more neighbours and less slots decrease the probability of transmission.

These probabilities are not very encouraging: under quite ordinary circumstances ($N_s = 8, m_i = 7$ for example), the success ratio is under 40%. Luckily, there are two mechanisms that enable robust networking: nodes will send a message every round, increasing the cumulative probabilities, and there is a technique called 'scheduling' for very dense neighbourhoods. These effects are further analysed in appendix B.

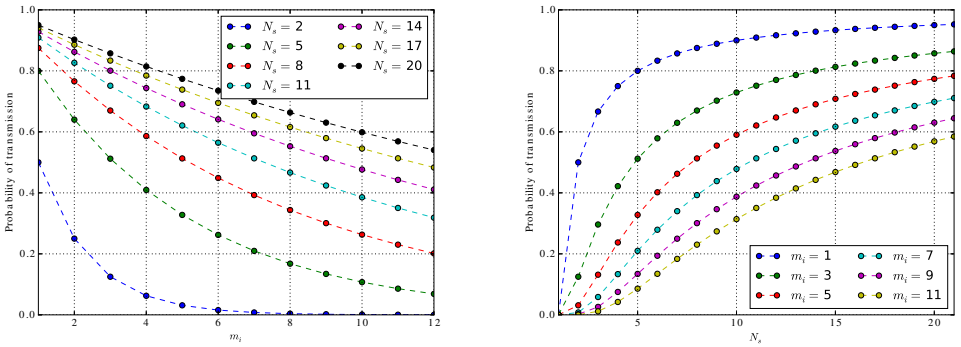


Figure 3-6: The probability of non-collision for different numbers of slots, and different neighbourhood degrees.

3-5-2 Comparison with Slotted ALOHA

Now that the probabilities are approximately known, we will try to compare throughputs. gMAC is inspired by the well-known Slotted ALOHA protocol [1, 55]. In that protocol, nodes use slots for transmission of messages, and every node has a certain probability of sending within a slot. This probability is Poisson distributed over the group of nodes: there is an average amount of messages transmitted per slot L, such that the probability of messages being 'on-air' X is exactly x in a slot is:

$$\Pr(X = x) = L^x \frac{e^{-L}}{x!}$$
$$\Pr(X = 1) = Le^{-L} = S_A$$

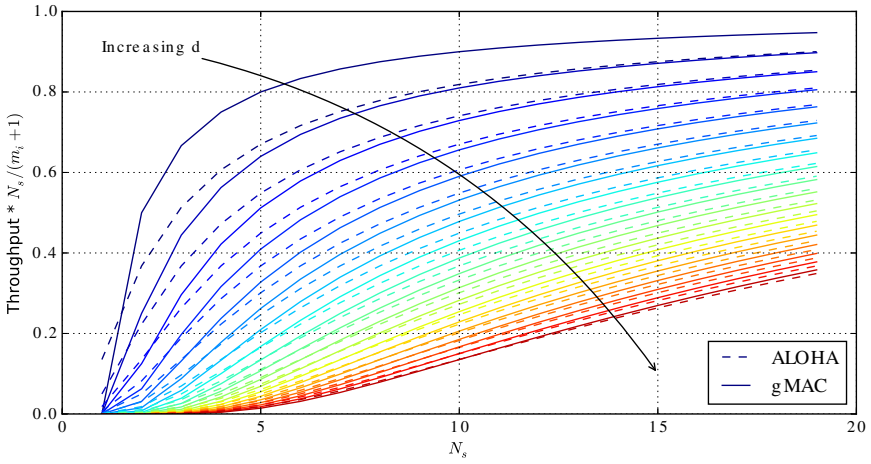
For $X = 1$ the transmission succeeds, since there is no collision. This probability is commonly interpreted as the throughput S: the fraction of slots that contain a successful transmission. This function has a maximum for $L = 1$, being $\frac{1}{e} \approx 0.36$. So under the most ideal circumstances, about 36% of the slots contain a successful transmission. Other slots are either empty (also 36%) or contain a collision (the remaining 28%).

L corresponds to the number of messages per slot, which would be $\frac{N}{N_s}$ in gMAC. There is the subtle but important difference however, that the number of messages sent per node is a stochastic variable in Slotted ALOHA, while it is a set number in gMAC (where only the moment of sending is randomized). For gMAC, the throughput (and thus the probability that exactly one message is sent in a given slot) is the probability of successful transmission, multiplied by the number of messages that might be sent $m + 1$, divided by N_s:

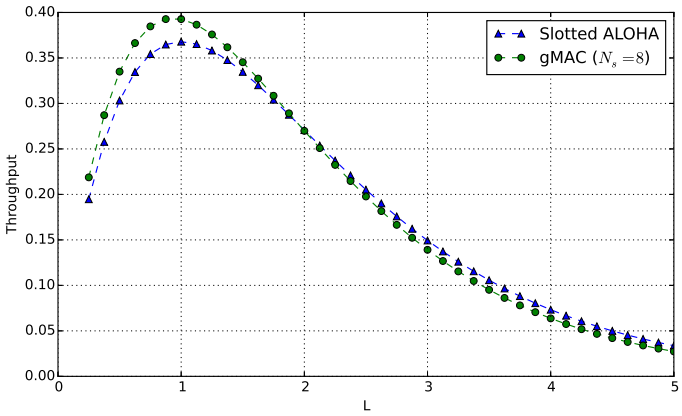
$$\text{gMAC: } \Pr(X = 1) = \frac{m_i + 1}{N_s} \cdot (1 - \frac{1}{N_s})^{m_i}$$
$$\text{Slotted ALOHA: } \Pr(X = 1) = Le^{-L} = \frac{m_i + 1}{N_s} \cdot e^{-\frac{m_i+1}{N_s}}$$

These functions are very similar, see Figure 3-7a. In the limits of m to infinity and N_s to infinity, the functions are equal, see appendix B-4.

In Figure 3-7b the throughput of both methods is compared for a common N_s, varying m to achieve a range of values for L. There is a slight difference between the curves: in sparser neighbourhoods gMAC performs better, and in dense neighbourhoods slotted ALOHA is a little more successful.



(a) The probabilities of success for gMAC and slotted ALOHA compared



(b) The throughput for gMAC and slotted ALOHA compared, $N_s = 8$, m varied to achieve L.

Figure 3-7: gMAC and slotted ALOHA compared on throughput.

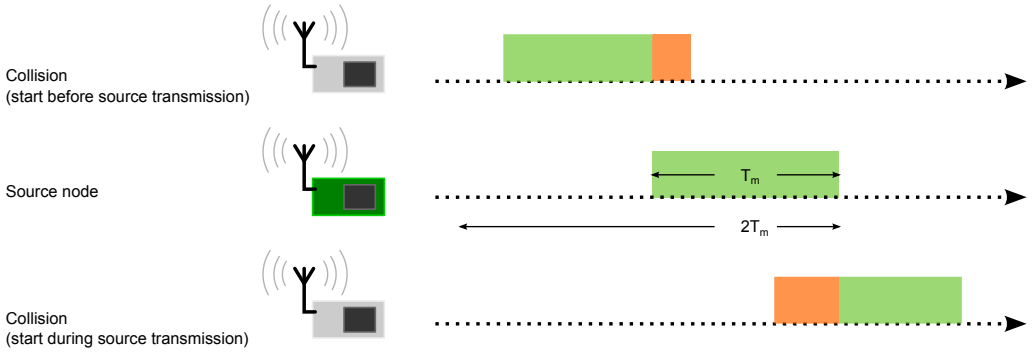


Figure 3-8: The modes of collision of pure ALOHA.

Alternative: Continuous gMAC

Although the throughput of gMAC seems on par with slotted ALOHA, it should be noted that a slot is not spent entirely on communicating. In fact, under default settings, when the usual 9 guard ticks are observed and there are 32 bytes per packet at 2Mbps (leading to 10 ticks transmission time), roughly two thirds of the slot is spent waiting. So compared with a perfect slot alignment, two thirds of the energy is wasted, drastically reducing the actual throughput. Pure ALOHA (the unslotted, original version) is only half as efficient, and does not require strict timing. Would a pure ALOHA yield better throughput?

To investigate this, we can devise a pure ALOHA-like variant of gMAC, called continuous gMAC. Slots are not used in communication, and every node picks a random clock tick to start transmitting. The time a transmission takes is T_m. Obviously it's useless to start a transmission less than T_m ticks before the end of the round, so for a fair comparison to normal gMAC there are $N_s(T_m + 2T_g) - T_m$ ticks (T_g being the guard time) to choose from. The complete active period T_a is $N_s(T_m + 2T_g)$.

The weakness of pure ALOHA is that whenever another node starts transmitting during the transmit time of the sending node, or had started transmitting less than the duration of a transmit before it, both transmissions fail. That is, the 'vulnerable period' during which no other node should send is two times the duration of sending, see Figure 3-8.

The chance of picking an arbitrary tick x as transmission tick T_i for node i is:

$$\Pr(T_i = x) = \frac{1}{N_s(T_m + 2T_g) - T_m}$$

In the case of a two node network, a collision occurs if the receiving node broadcasts in the $2T_m$ period indicated in Figure 3-8. If the start of the transmission T_i happens to be in the first T_m ticks, this probability is lower, since the receiving node is not allowed to send before the active period begins. Since this will lead to a different probability for every possible choice of $T_i < T_m$, this will complicate the analysis. We adopt the conservative assumption that the probability of collision remains the same in the whole active period. As the fraction of message length to active period duration ($\frac{T_m}{T_a}$) becomes larger, this assumption will become more restrictive.

In this scenario, the probability that the receiving neighbour sends in one of ticks in the $2T_m$ period is:

$$\Pr(T_i \in 2T_m) = \frac{2T_m}{N_s(T_m + 2T_g) - T_m}$$

This is the chance of collision between two nodes.

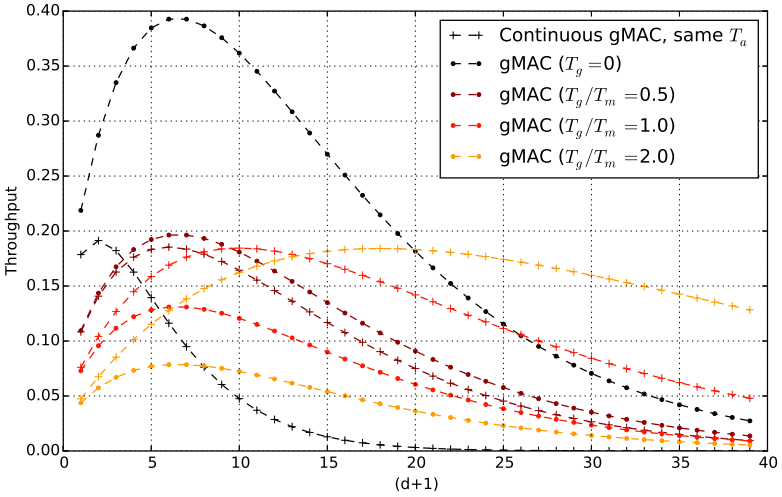


Figure 3-9: The throughput per tick instead of per slot, comparing gMac with its pure ALOHA counterpart: continuous gMac. With increasing guard times, gMac becomes less efficient.

When there are m neighbours, the chance that none of them sends in a given period of $2T_m$ slots is:

$$\Pr(T_i \notin 2T_m \forall i \in N) = \left(1 - \frac{2T_m}{N_s(T_m + 2T_g) - T_m}\right)^m$$

To get to the throughput, we should normalize this probability again. Because slots are a meaningless concept to continuous gMAC, we want to compare on the metric 'successful messages per energy spent'. If we multiply the probability of successful transmission with $\frac{(m_i+1)*T_m}{N_s(T_m+2T_g)}$ (number of ticks spent on sending divided by total number of ticks) for both cases, we arrive at a comparable number.

The results are shown in Figure 3-9. In the hypothetical scenario without guard times, gMAC is clearly more efficient. When the guard times are half the time taken by message transmission (so half the slot is filled with guard times), gMAC is still slightly better than the continuous version over the whole range of densities. When the guard times become more than half of the slot, gMAC loses against continuous gMAC in throughput, and continuous gMAC maintains a maximum around $\frac{1}{2e} \approx 0.18$.

The guard times serve a goal of course: when synchronization errors are present, the guard times prevent loss of throughput. In a continuous version of gMAC however, the loss of throughput due to synchronization errors is caused by the decreased overlap of active periods only. In the normal, slotted gMAC, every slot loses throughput (via a lower transmission success probability) when a synchronization error is present. So we expect that even in the presence of synchronization errors, the results from Figure 3-9 hold up.

Designing a new MAC protocol is not the aim of the thesis. Continuous gMAC might be a good principle in some situations, but there are many details and pitfalls that are yet to be further investigated before drawing rash conclusions. What we can conclude from the preceding analysis however, is that preventing synchronization errors is a crucial part of the current MAC protocol. A synchronization protocol should limit the maximum synchronization error to at most half the time needed for sending a message, or else using gMAC is a very suboptimal decision!

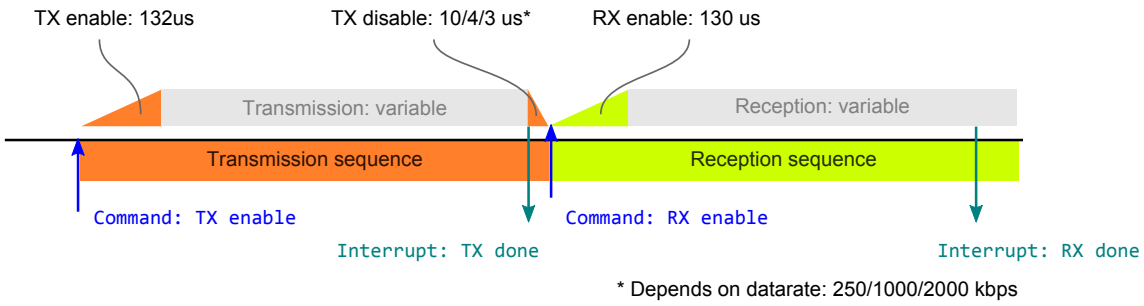


Figure 3-10: The timing of the sequences to transmit and receive messages. Source: [46]

Table 3-2: The maximum shortening of a slot under different system settings. Lifetimes are estimates from Chess Wise's energy balance calculator.

Package size	Datarate	Length of a slot [ticks] $T_g = 9$	$T_g = 1$	Decrease	Typical lifetime increase
32 bytes	2 Mbps	27.65	11.65	58%	54%
32 bytes	1 Mbps	33.04	17.04	48%	43%
32 bytes	0.25 Mbps	65.38	49.38	24%	20%
64 bytes	2 Mbps	31.84	15.84	50%	45%
64 bytes	1 Mbps	41.43	25.43	39%	33%
64 bytes	0.25 Mbps	98.94	82.94	16%	13%

3-6 Lifetime Increase

In the previous section the influence of synchronization on throughput was sketched, in this section we will quantify the lifetime increase that is possible with better synchronization algorithms. The measure of interest here is the ratio between guard time and message transmission time, as demonstrated in the previous section. The transmission time is largely dependent on the message size, and can be determined via the following formula:

$$\begin{aligned} \text{Time on air} &= \frac{\text{bits to be sent}}{\text{datarate [bits per second]}} \\ &= \frac{\text{preamble} + \text{message size [bits]}}{\text{datarate [bits per second]}} \\ \text{Transmission time} &= \text{Time on air} + \text{Radio enable time} \end{aligned}$$

The radio TX enable time is 132 µs, so roughly 5 ticks. This time is spent on ramping up the radio. Switching back from TX to RX takes 130 µs for the same reason, see Figure 3-10 for an overview. If we want nodes to transmit and receive successfully in consecutive slots, we will need to account for this switching in the transmission time. Because the oscillators have a slightly different phase, the best attainable result would be to drive the guard time to 1 tick.

The increase in energy efficiency, and thus lifetime, is attained by shortening the length of a slot, thus shortening the active (energy consuming) period. The exact amount of saving depends on system settings, as displayed in Table 3-2. The most common setting would yield the highest savings, but even in a more conservative setting the gains are significant. The increase in lifetime (with a frame length of 1 second) is somewhat lower because there is a constant overhead energy use. It is obvious, however, that a tighter synchronization could yield very big benefits.

Chapter 4

Algorithms

The main part of this chapter, section 4-2, gives an overview of synchronization algorithms that can be found in literature. Since the aim is to find algorithms and approaches that can be applied to MyriaNed, we will first outline the requirements in section 4-1. Finally, the possibility of using a temperature measurement for improving synchronization is investigated in section 4-3.

4-1 Algorithm Requirements

In chapter 3, it has become clear that the MAC protocol of MyriaNed requires a synchronization algorithm to take a set of phase difference measurements as an input, and produce a correction of the idle time as an output. These are used to reduce the phase differences in the next round. Many algorithms try to compute a translation of the local clock value τ to arrive as close as possible to the reference t (the synchronization versus synchronicity distinction, see section 2-3). Although it presents an extra step, it is possible to use an approximate clock to compute a better idle time.

The features of the network imply more demanding requirements. Because an essential property of MyriaNed is that there can be no single point of failure, this should be respected by an algorithm. Thus schemes where a single node is appointed as a master clock are excluded. The large diversity of network deployments means that topology should not be critical to the algorithm. In any wireless network links are unreliable, but algorithms should also be able to cope with dynamic networks (moving nodes) and the extremes of density and sparsity. Moreover, additional nodes should be able to start up, join the network and become synchronized at any moment. Ideally, a synchronization algorithm has only a single mode of operation (as opposed to solutions where first an initialization phase has to be performed before other processes can start).

Of course, a protocol should fit the hardware (e.g. clock granularity), and achieve an accuracy that is useful to us: maxima of hundreds or preferably tens of microseconds.

Since the overarching goal is to improve energy efficiency, the overhead of algorithms in terms of computations and extra message content should be minimal. As noted by Elson [16], there is no single best algorithm: each algorithm occupies a part of the solution space, and represents a trade-off between resource consumption and performance. The difficulty of picking or designing a suitable algorithm lies in finding the most efficient trade-off while attaining the performance needed. Since experimental results are rarely

available (and even then those results never mention energy consumption) an objective optimization can not be done. Still, we might keep this trade-off in mind when considering algorithm candidates.

4-2 Literature

Achieving synchronization is not a problem unique to MyriaNed. Synchronization of clocks in distributed systems has been a topic of academic interest since the 70's, see for example the classic [36]. With the advent of wireless sensor networks – posing their own set of challenges – scientists have shown renewed interest [65, 16, 60]. A literature study was done to survey the solutions and algorithms previously published, and an overview will be reproduced here. Algorithms typically differ on three measures:

1. The goal definition: what is meant by synchronization, and when a synchronization is reached.
2. The network properties: what assumptions can be done about the nodes and links, what information is available to the nodes.
3. The approach of the researchers: the reasoning behind the model, and how the proper functioning can be demonstrated.

The first two measures were discussed in chapter 2. The variance in the third, very subjective measure will be outlined here.

The problem of synchronizing wired networks is traditionally cast as a parameter estimation problem. The synchronization problem is stated in the unknown clock drifts and message delays, which are to be estimated. Strategies for attaining a Maximum Likelihood or Minimum Variance estimate were discovered. These algorithms always operate in a point-to-point fashion: two nodes enter a synchronization procedure, and after sending messages back and forth, come out synchronized to a certain degree. This scheme can involve a central reference to get external synchronization. The Network Time Protocol (NTP) is the exponent of this class. Because of their point-to-point and centralized nature, and assumptions (about link certainty for example) they work fine for a wired network, but are inefficient or even impossible to employ on a wireless network.

This was recognized by the Computer Science community when the first practical WSNs were developed [16]. The challenge sparked several proposals that approached the problem in a pragmatical way, but most still relied on the traditional concepts of reference nodes and point-to-point synchronization. Some of these algorithms are quite successful, and remain in use and the performance benchmark today (most notably FTSP).

Control scientists proposed new algorithms shortly after, relying on more distributed techniques. From distributed control concepts such as consensus protocols were known, and could be applied to this problem. By taking the perspective of individual nodes trying to infer their own state from noisy measurements, or by viewing the time keeping in the network as one large switching dynamical system, new roads to synchronization were discovered.

Quite independently from all this, the topic of oscillator synchronization unrelated to WSNs is a mathematical field of study with a long history. The Kuramoto model for coupled nonlinear oscillators was proposed in 1975, and has received widespread attention [65]. With the interest of mathematicians in complex networks starting at the end of the century, the interest in oscillator coupling revived. The synchronized behaviour of fireflies,

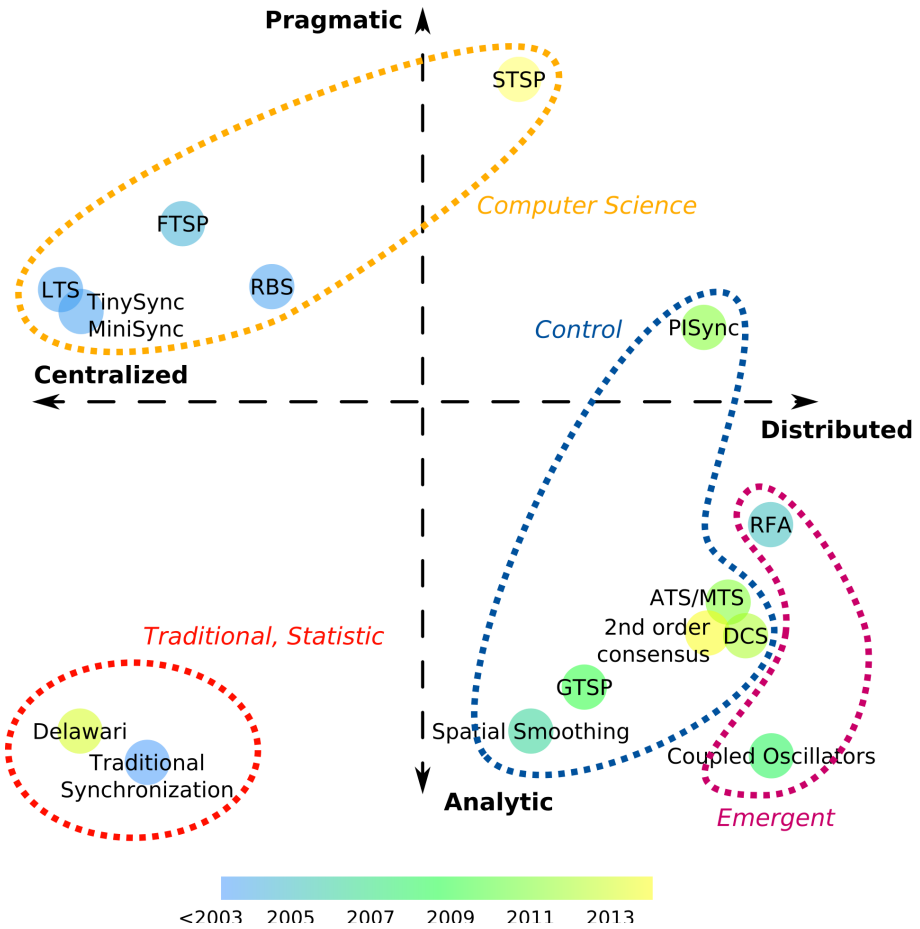


Figure 4-1: A collection of synchronization algorithms from literature roughly positioned on two axes: the degree of distributedness, and the pragmatism vs. analysis leading up to the algorithm. The different approaches are grouped together.

heart muscles and firing neurons in the brain all have been successfully modelled using these techniques. This field of study seeks to understand emergent behaviour of large networks. These algorithms commonly require phase information of individual oscillators, which is not available in the MyriaNed hardware.

In Figure 4-1, the different algorithms are plotted along two axes: their distributedness and the approach taken in their development (analytic or pragmatic). Centralized algorithms rely on a reference node, or construct clusters of local time. We demand a fully distributed algorithm, in which every node fulfils the same role. Pragmatic algorithms are predominantly early solutions that are constructed heuristically, focussing on specific hardware issues, and supported by experiments. Analytical algorithms have a mathematical basis, and analytic substantiation of their performance claims. They often need translation to a hardware platform before they can be deployed.

The current Median algorithm essentially falls in the upper right corner of the field, being fully distributed but mostly pragmatic in nature. The aim of this graduation project is to improve the MyriaNed solution by drawing inspiration from the more analytic algorithms, while retaining its distributed nature.

In the following, algorithms from each category will be briefly described, and their merits discussed.

Pragmatic and Traditional Approaches

Since we aim for a fully distributed (and therefore scalable and robust-to-dynamics) algorithm, the traditional and pragmatic approaches are mostly not very suitable. They represent important views on the synchronization problem however, and for completeness we will mention some of the landmarks of these approaches here.

In the traditional approach, applied to a Wireless Sensor Network (WSN) in [12, 59, 74], algorithms revolve around the pairwise synchronization of nodes. Three communication models are commonly distinguished:

- *Sender-receiver:* in the most basic model, two nodes exchange messages: node A sends, node B replies, etc. Given the broadcast nature of a wireless medium, this is rather inefficient. The following two models aim to resolve this.
- *Receiver-only:* in this model, one node is considered the master node, and sends messages. Other nodes receive these messages and synchronize to the master clock. Note that the master node never receives messages nor adjusts its clock.
- *Receiver-Receiver:* a hybrid between the two: again one node is a sender, but this time the nodes receiving its messages note the time of reception. By exchanging these perceived times, they synchronize to each other.

Remarkably, none of the three can describe the broadcast-and-local-correction model MyriaNed uses. Receiver-receiver is a broadcast protocol, but in MyriaNed it is the receiver that should be synchronized to the sender, not the receivers to each other.

Reference Broadcast Synchronization (RBS) One of the first algorithms tailored to the needs of synchronization in Wireless Sensor Networks is RBS [16]. The concept is a receiver-receiver synchronization: one node sends a broadcast message, and other nodes note the time of reception. Since this moment will be almost equal for all nodes, they can then exchange their perceived times at the moment of the reference broadcast to know their time differences. There are three drawbacks to this approach. Firstly, the time difference information is used to construct a translation table, which is unfeasible for large scale and/or dynamic networks. Secondly, the node sending a reference broadcast does not participate in the ensuing synchronization, so the procedure must be repeated, consuming resources unnecessarily. And finally, the communication overhead of comparing and communicating all the pairwise differences in reception time is quite large.

Other Reference Node and Pairwise algorithms Many popular schemes for synchronization involve the election or definition of a reference node, which is supposed to keep the time that all other nodes try to follow: a centralized approach. The most notable in this class is the Flooding Time Synchronization Protocol (FTSP) [40], the default synchronization protocol in TinyOS [39]. Other algorithms using this approach are Delay Measurement Time Synchronization Protocol (DMTS, [52]), Lightweight Time Synchronization (LTS, [69]), Time Diffusion Synchronization (TDS, [66]) and Timing-sync Protocol for Sensor Networks (TPSN, [22]), and the algorithms proposed in [60, 38, 67, 12]. Although the performance is experimentally shown to be good, the construction of a network-wide synchronization takes considerable time, and the network becomes vulnerable to node failure (of the reference node or a node in a critical path in the tree). What's more, two neighbouring nodes stand the chance of being in two different branches of the spanning tree, making their synchronization possibly very poor due to synchronization errors accumulating per hop [64].

Control Approaches

A concept in WSNs that garnered interest over the last decade is consensus. The consensus problem is simple: all nodes in a network need to agree on a value. Under weak assumptions (eventual or periodic connectedness), the convergence of a very simple average-over-all-neighbours algorithm can be proven (see chapter 5). This convergence property, combined with the fully distributed and broadcast-based nature makes it an attractive avenue for synchronization research. Some exponents:

Average TimeSync (ATS) Average TimeSync (ATS) [57] is an algorithm with a cascaded consensus architecture. Based on a broadcast scheme, every node estimates its relative drift and offset towards all its neighbours. A drift estimate and an offset estimate with regards to a virtual reference clock is communicated between nodes. It is this virtual reference clock about which consensus needs to be reached. The exact value of the reference clock depends on initial conditions of the nodes, and the communication topology. The algorithm is proven to converge for topologies similar to those generated by gMAC, using positive products of stochastic matrices and a Lyapunov function. A conservative lower bound on convergence is provided. The algorithm is shown to be on par with FTSP performance-wise, but more robust.

In reaction, He et al. [26, 27, 28] proposed Maximum Time Synchronization (MTS), operating along the same lines but always picking the maximum instead of a weighted average to reach consensus. The algorithm is proven and shown experimentally to converge faster than ATS. In [27] a comparison with DCS (an algorithm from [11]) is made for networks with probabilistic links. There is one crucial drawback: since the maximum is a biased operation, the network will continuously speed up, making it infeasible in the long run. This can be counteracted by including a minimum as well, but this increases the communication overhead.

Gradient Time Synchronization Protocol (GTSP) In [64], GTSP is proposed. The resulting algorithm is very similar to ATS, and has a consensus-like logic: an internal state is updated regularly by taking the average of all the neighbours' states. The theoretical background to this solution, mostly drawn from [18], is completely different. In that paper Fan and Lynch [18] introduce the Gradient Clock Synchronization problem. Noting that minimizing differences between neighbours is the most important goal in synchronization for e.g. TDMA schedules, they formalize this into a gradient of time differences over the network that is to be minimized. Using a timed automata model, a lower bound on the time difference between nodes that depends on the network diameter is derived. Interestingly, this would mean that the diameter of the network as a whole limits the performance of an algorithm on a pairwise basis. The assumptions made do not hold up very well for MyriaNed: a crucial part in the formal analysis is the 'hiding' of time differences by varying message delays. Since the message delays are known and constant in MyriaNed, this is not applicable. Nonetheless, the Gradient Problem is very close to our problem statement, and the timed automata present an interesting alternative perspective.

Model-Based Clock Synchronization (MBCS) In [20], an advanced clock model is used as a basis for a synchronization algorithm: Model-Based Clock Synchronization (MBCS). The proposed system model leads to an optimal filter for estimation of parameters over links. This Kalman-Bucy filter is not very practical: it has trouble with the asynchrony in the network, the distribution of computations and discretization, and requires integration with respect to the (unknown) reference time. These hurdles are overcome by the authors to arrive at a suboptimal, distributed solution that can be enhanced by the Spatial

Smoothing method. The clock model is broad enough to be applicable to our network, but perhaps too broad to assume that a model-optimal approach will be optimal in reality.

Spatial Smoothing Also known as Distributed Time Synchronization Protocol (DTSC). In the same group as Freris et al. [20, 21], a distributed protocol was proposed by Solis et al. [63]. Starting from a pairwise scenario, parameters are estimated using a linear regression over exchanges. It is noted that for any cycle in the network, the offsets (and the logarithms of the drifts) should add up to zero. Via a quadratic optimization the optimal adjustment of estimates to enforce these constraints is calculated. Via a smart distribution of computations, this optimization leads to a consensus-like protocol of averaging the neighbours' estimates, as also noted by Schenato and Fiorentin [57].

This protocol is designed for a bidirectional, symmetric network, which is strongly connected. The clocks are linear, but the analysis is done only for driftless (only offset) clocks (with hints on how to do it for drift as well). These assumptions are still quite far removed from MyriaNed, and no experimental results are reported.

PISync The research group behind Carli and Zampieri [7], Carli et al. [9] has been viewing the problem from the nodes' point of view, and proposed an proportional-integral (PI) controller for various settings. Using the time differences with neighbours recorded after reception of a message, nodes correct their clocks. A proportional correction is done to reduce the clock offset, and an integral part is used to diminish the frequency differences. Since all nodes communicate their own state only, and agree on a common virtual clock, the dynamics of the algorithm are again very much like a consensus protocol.

For known and fixed topologies, the choice of gains can be formulated as a convex optimization problem, shown in [8]. In [7], the inherent slight asynchrony of the algorithm execution is tackled for a pseudo-synchronous scheme: nodes update their clocks after a round in which (almost) all neighbours are heard – exactly what MyriaNed does! In [75] (by the authors of ATS and PISync as well) a pragmatic version with error-dependent integral gain is used in hardware experiments, and compared to benchmark algorithms FTSP and GTSP, with good results.

Emergent Behaviour Approaches

The spontaneous synchronization of coupled oscillators is a phenomenon that has fascinated mathematicians in recent years [65, 47]. The behaviour of swarms of birds and insects, schools of fish or flocks of animals is modelled and imitated. The distributed characteristic of WSNs, and the need for simple local behaviour that should spark intelligent global behaviour has led researchers to look at the results from these studies for solutions.

Reachback Firefly Algorithm (RFA) In the context of synchronization protocols, this has led to RFA[72], inspired by synchronously flashing swarms of fireflies. The algorithm works by having every node 'flash' (broadcast) periodically. If a flash is received by a node at a different time than its own flashing, the duration before its own next flash is shortened. The amount of shortening is subject of a 'firing function'. Following the work of Mirollo and Strogatz [42], convergence of the algorithm is proven under assumptions on the firing function that are easily met. The implementation of the algorithm prompts adjustments that limit this convergence however, and the experimental performance is not outstanding. Although the concept of the paper is novel, the algorithm does not use drift estimation, limiting its accuracy in the long run, and moreover it demands that all nodes listen continuously, which is not acceptable due to energy consumption.

Other pulse-coupled methods In [62], the synchronization problem is viewed from a similar perspective of pulse-coupled oscillators. The overview starts out with continuous-time sine waves that are coupled by interchanging pulses, a process controlled by the physical layer. After the transition to discrete time, the problem is similar to the gMAC synchronization, since pulses do not contain time information (like the MyriaNed packets), and the time difference has to be inferred. Aside from the integrate-and-fire methods (like RFA) that Simeone et al. [62] admit are hard to adjust for practical implementation, and leave little degrees of freedom for tuning, the theory of Phase-Locked Loops (PLLs) is mentioned. In [61], the WSN synchronization problem is framed in terms of PLLs. The proposed solution demands access to the physical layer however, which we do not have.

4-2-1 Conclusion

As demonstrated in this chapter, there is no shortage of synchronization protocols. A technique that is commonly used but not yet employed in MyriaNed is to find a way of estimating a node's own drift, and compensate for that in the long run.

Within the confines of MyriaNed, very few algorithms can be applied however. The approach taken by control scientists seems most promising, since it yields fully distributed algorithms with an analytical basis. Most interesting are ATS and PI-based methods, notably PISync. In chapter 6, two algorithms will be adjusted to be applicable to MyriaNed, accompanied by an algorithm which we devised ourselves.

4-3 Temperature

Clearly the largest influence on oscillator drift is temperature. The current hardware platform – the Nordic nRF51 – is equipped with a temperature sensor. Although this sensor has very limited accuracy, it might be useful for correcting a node's idle time as a function of temperature. In this section we investigate whether it is indeed an improvement to use this temperature signal in a feed forward correction.

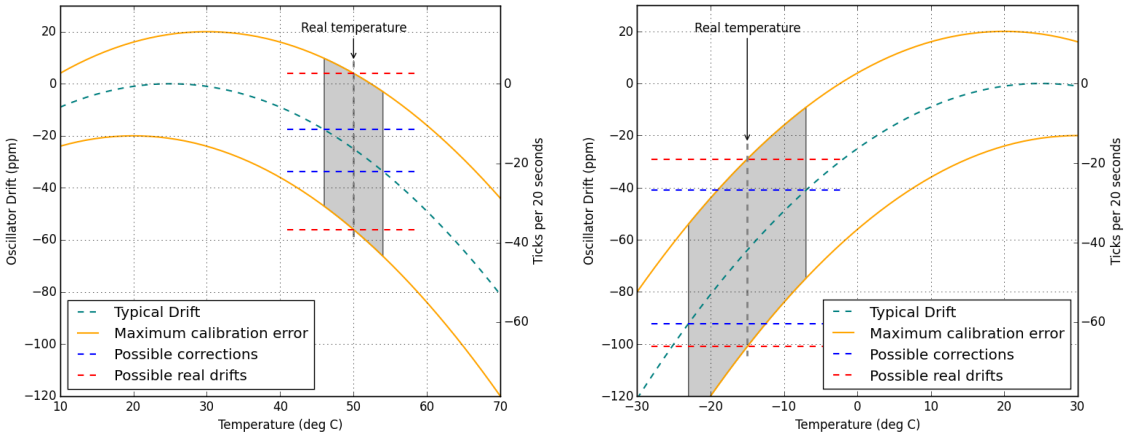
4-3-1 Measuring Temperature

The internal temperature sensor of the Nordic nRF51[45] has a range of $-25\,^{\circ}\mathrm{C}$ to $75\,^{\circ}\mathrm{C}$. From $0\,^{\circ}\mathrm{C}$ to $60\,^{\circ}\mathrm{C}$ it has an accuracy of $\pm\,4\,^{\circ}\mathrm{C}$, outside that center range it is $\pm\,8\,^{\circ}\mathrm{C}$. The resolution is $0.25\,^{\circ}\mathrm{C}$. During operation, the sensor draws $185\,\mu\mathrm{A}$, and one measurement takes $35\,\mu\mathrm{s}$, so one measurement takes a negligible amount of energy. On many hardware platforms, an external temperature sensor is available that provides a better accuracy. We will work with minimal means first.

4-3-2 Correction

Typically, 32 kHz quartz oscillators have a calibration offset of $\pm 20\,\mathrm{ppm}$[25], and a temperature coefficient of $-0.04\,\mathrm{ppm}\,^{\circ}\mathrm{C}^{-2}$. The turnover temperature lies somewhere between $20\,^{\circ}\mathrm{C}$ and $30\,^{\circ}\mathrm{C}$.

Using the specifications of the temperature sensor and the oscillator, we can assess the validity of a drift estimate if we used a measured temperature value and a typical oscillator characteristic, see Figure 4-2. The range of values that a node might infer from its temperature measurement (in blue), assuming that it has a typical oscillator, are always well within the range that the oscillator drift can have given the variability in calibration



(a) For a real temperature of 50 °C

(b) For a real temperature of −15 °C

Figure 4-2: The shaded areas indicate the uncertainty regions, taking into account the inaccuracy of the temperature measurement and the oscillator. The blue lines indicate the range of values that can result from a calculation assuming a typical oscillator and a measurement from the temperature sensor. The red lines indicate the range of values that an oscillator can have given the temperature and variability between oscillators.

(in red). In other words: the blue range is always within the red range, and thus it is never an impossible estimate.

In order to see whether using this estimation is always better than doing nothing, another comparison is made. For both actions, we compute the worst case error (the real drift is removed as far as possible from the estimate) and the best case error (the real drift is as close as possible to the estimate), see Figure 4-3. Some observations:

- The worst case estimate with temperature compensation is always better than the worst case zero estimate.
- The worst case estimate with temperature compensation is always worse than the best-case of a zero estimate.
- Below 0 °C and above 50 °C, the zero estimate is guaranteed to be wrong.

Although the second statement seems to be worrisome, it is a matter of statistics. The average error between best and worst case is always considerably lower for the temperature-based estimate than for the zero estimate. Therefore, doing a feed forward compensation of the measured temperature in the synchronization algorithm might make the synchronization more robust to temperature changes. This feed forward compensation is evaluated with experiments, which are treated in subsection 8-3-3.

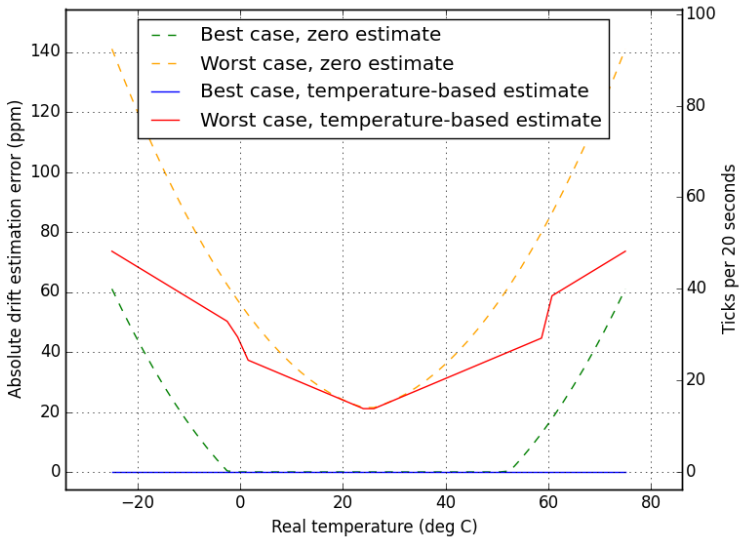


Figure 4-3: The best and worst case drift estimates when using temperature or estimating 0.

Chapter 5

Stability

In previous chapters, the essence of the synchronization problem has been laid out. The requirements posed by the hardware and software context for possible algorithms have been clarified, and solutions proposed in literature are surveyed. The algorithms executed by nodes individually are often fairly simple from a control point of view. Nevertheless, providing guarantees of stability of the algorithm executed over a whole network proves to be less trivial. Since the integrated disturbance caused by different drift factors is unbounded, controllers that are stable for a single node may lead to dissynchronization in the network. In this chapter, we will introduce the mathematical model of synchronization algorithms in MyriaNed, which can be framed as a consensus process. Methods to prove stability of synchronization algorithms will be introduced, and a proof of stability for a class of synchronization algorithms will be given.

5-1 Modelling

5-1-1 Clocks

Each node in the network keeps its local time using a clock. As noted before in chapter 2, these clocks are commonly modelled as linear clocks:

$$\tau_i(t) = a_i t + b_i$$

Where the local clock value τ_i of node i, is expressed as a function of reference time t. The reference time, sometimes called a wall clock time, is the same for all nodes[1]. The local clock value, which is a measurement of time in a sense, is distorted by drift factor a_i and offset b_i. The drift factor is the deviation of the oscillator's frequency from the nominal frequency, usually expressed as parts per million (ppm). We'll denote the nominal frequency as f_n, and the actual frequency as $f_i(t)$. The difference is generally small, and a_i will be close to 1. The drift is partly constant due to miscalibration, and partially time-varying due to external influences and ageing. It is assumed to change very slowly – allowing us to model it as a constant. The offset is caused by different startup times of clocks, and is constant. The three variables τ_i, t and b_i should be in the same unit of time, usually seconds.

[1] We'll ignore effects of light speed and relativity. Light travels about 300 meters in a microsecond. The transmission range of nodes is usually smaller than that. Moreover, the accuracy that we're striving to achieve (tens of microseconds) is one order of magnitude lower.

The gMAC protocol in which the algorithms under consideration will be executed operates in rounds. To make the considerations in this chapter independent of round time T, and to get rid of clock values growing unbounded over time, we can express the clock values of nodes as the difference with the reference time for that round. We subtract t from the clock value, and get a phase offset for every node:

$$\begin{aligned} x_i(t) &= \tau_i(t) - t = (a_i - 1)t + b_i \\ \text{Discretize into rounds: } & t = kT \\ x_i(k) &= (\tau_i(kT) - kT) \\ &= (a_i - 1)kT + b_i \\ x_i(k+1) &= (a_i - 1)kT + b_i + (a_i - 1)T \\ &= x_i(k) + T(a_i - 1) \end{aligned}$$

Where $x_i(k)$ is the phase offset of node i at the very start of round k. A requirement for the validity of this description is that synchronization errors do not grow so large that the active periods of different rounds start to overlap, and communication between rounds happens. Long before this unmodelled processes like the join mechanism start working however, so this is of no concern.

If we choose a derived drift value $d_i = T(a_i - 1) \approx 0$ for this description, and realize that $b_i = x_i(0)$ the evolution of clocks in a network of N nodes can be expressed as a linear system:

$$x(k) = \begin{pmatrix} x_1(k) \\ \vdots \\ x_N(k) \end{pmatrix} \quad d = \begin{pmatrix} d_1 \\ \vdots \\ d_N \end{pmatrix}$$

$$x(k+1) = x(k) + d, \quad x(0) = b \tag{5-1}$$

Note that the system matrix is I (identity), so the eigenvalues are all 1, with orthogonal unit eigenvectors, representing the ideal clocks. The vector d can be seen as a disturbance acting on the system, which is bounded and small. In reality, d contains a stochastic component as well [70]. The system without disturbance is marginally stable (a discrete-time system with eigenvalues of magnitude 1), so d acts as a destabilizing force.

To stabilize the system, information must be exchanged between nodes, coupling the states. This network model will be introduced in the next section.

5-1-2 Network

A convenient way to model wireless mesh networks is by describing the network as a graph. The graph of a wireless mesh network consists of a set of N nodes or vertices $\mathcal{V}= \{1, \ldots, N\}$, and a set of edges $\mathcal{E}$, consisting of (i, j) pairs to denote communication from node i to node j. Note that the graph of the network is directed, so the existence of (i, j) does not imply (j, i). When a node i receives messages from another node j, j is said to be a neighbour of i. The set of neighbours for a node i is $\mathcal{N}_i$.

We must consider dynamic networks however, and under gMAC many sent messages will fail. We therefore define a switching network topology, where $\mathcal{E}(k)$ is the set of successful connections in round k. This edge list can be turned into an adjacency matrix $A(k)$ for

every round:

$$A(k) = \begin{bmatrix} a_{11}(k) & \cdots & a_{1N}(k) \\ \vdots & \ddots & \vdots \\ a_{N1}(k) & \cdots & a_{NN}(k) \end{bmatrix}$$
$$w_{ij}(k) = \begin{cases} 1, & (j,i) \in \mathcal{E}(k) \\ 0, & \text{else} \end{cases}$$

The links are indicated by a weight w_{ij}. As a matter of convention, the network does not contain self-links. That is, (i,i) can not occur in $\mathcal{E}$ and the diagonal entries of A are 0. Because the network is directed, $w_{ij}(k)$ does not equal $w_{ji}(k)$ in general. In some network representations w_{ij} can be any value, and is used to indicate the weight of a link. We have opted to make w_{ij} a binary variable describing the possibility of communication. Apart from this physical property, an algorithm can choose to weigh or even ignore links. This will be indicated later on by (variations of) a_{ij} and b_{ij}, which are related to w_{ij} but not identical.

5-1-3 Synchronization Action

With a network and node model in place, the missing piece is the synchronization algorithm. A synchronization algorithm should turn a collection of phase differences $\{(x_i(k) - x_j(k))|(i,j) \in \mathcal{E}(k)\}$ into a single correction of the idle time $\epsilon_i(k)$ for this round. Of course it is possible for a synchronization algorithm to maintain some internal state. In general:

$$\begin{aligned} \epsilon_i(k) &= f(\{(x_j(k) - x_i(k)) : j \in \mathcal{N}_i(k)\}) \\ &= f(\theta_i(k)) \end{aligned}$$

Note that a node i knows phase differences with nodes from which it has received a message, thus the links (j,i) towards i are of importance. For ease of notation, we define the set of differences with neighbours available to node i at time k as $\theta_i(k)$

In Figure 5-1 the position of the synchronization controller is clarified using a block scheme representing one round of a four node network. A single node i takes a set of phase differences with neighbours as an input, and produces a phase difference for the next round. This path goes via the controller, which specifies how the idle time should be adjusted (by a correction $\epsilon_i(k)$), and the clock that counts this idle time and is disturbed additively by $d_i(k)$. Depending on the network topology for the next round, the resulting phase $x_i(k)$ is subtracted from zero or more phases from other nodes in the network. All these phase differences are collected in a set, and these present the input for the next round. If no other nodes are heard, the set of phase differences will be empty, but the controller could still propose an $\epsilon_i(k)$.

Augmenting the complete system model from Equation 5-1 with the controller, we now have:

$$x(k+1) = x(k) + \epsilon(k) + d, \quad x(0) = b \tag{5-2}$$
$$\epsilon(k) = \begin{pmatrix} \vdots \\ \epsilon_i(k) \\ \vdots \end{pmatrix}$$

The block diagram visualisation barely works for four nodes: the network part is already quite cluttered, and intuitively representing an operation that combines several scalars into sets of scalars that are different in every time step is difficult. For the network scales that are of interest to us, this approach is impractical. It has provided some insight into the interaction between local (node-level) systems and the global scale however.

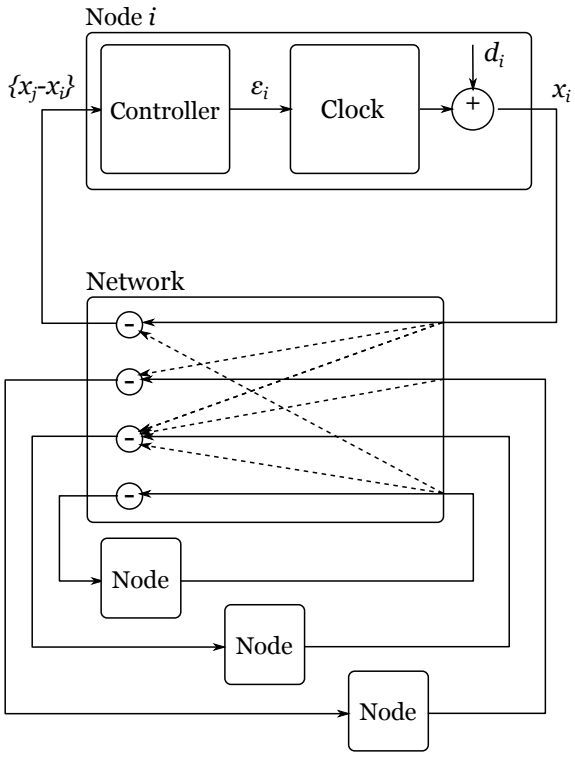


Figure 5-1: Framing the synchronization problem as a classical control problem using a block diagram. The block diagram represents a single round in a four node network. Node i receives one other node and is received by two others, but this changes every round. Each node has the same internal structure but a different disturbance d_i and a different initial condition. The local plants (nodes) are easily expressed, but for larger networks it is impossible to generate a sound block diagram.

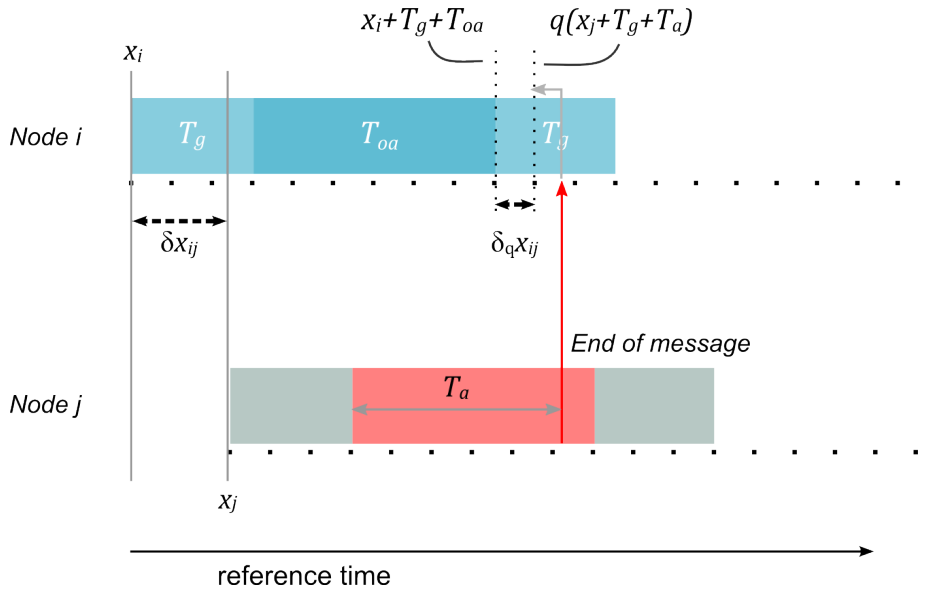


Figure 5-2: An inference of the time difference $(x_j - x_i)$ between node i and j, by node i.

5-1-4 Quantization

As briefly touched upon in chapter 3, the phase differences that nodes can use for synchronization are not calculated by exchanging timestamps, but inferred based on the arrival time of a message. The timestamping and triggering of these events is done using the internal 32 kHz crystal. Because the granularity of this crystal is rather large compared to the time scale involved, this mechanism introduces several distortions to the phase difference estimation. Additionaly, there is a deterministic misestimation of parameters.

If we perceive phase offset of a node with respect to reference time as a continuous variable x_i, the difference between two phases x_i and x_j can be any real value. There are three effects that disturb the estimation of this number:

1. Because the oscillators provide this phase only in discrete ticks, the estimation of $x_j - x_i$ is a round number of ticks as well. These ticks are large compared to the difference to be estimated: in simulations with one second frames and worst case drift conditions, phase differences are between -10 and 10 ticks.

2. The time taken for transmission of a message is saved and used in computations as a round number of ticks, while it might take a fraction of a tick more or less.

3. The interrupt mechanism in the processor, and the time it takes to do calculations is not completely deterministic: the timestamping of the receive moment can vary stochastically. This variation has experimentally been shown [35] to be less than a microsecond; about an order of magnitude lower than the accuracy we are trying to achieve.

The quantization has the largest influence on the estimation.

In Figure 5-2, the process of estimating the time difference between i and j, by node i is schematically drawn. The block shown is one slot, and the side bands of the block indicate guard times. The real time difference $x_j - x_i$ is indicated by δx_{ij}. Node j starts sending its message after the guard ticks in its slot. Some time T_a later (not a round number of ticks per se), the message is finished and has arrived at i, upon which the interrupt is raised and node i timestamps the arrival. This timestamping is a rounded down number of ticks. Let $q(x) = \lfloor x \cdot f_n \rfloor / f_n$ be the quantization operator. This operation rounds the

time down to the nearest number of ticks. The timestamp of arrival is $q(x_j + T_g + T_a)$. The expected time of arrival is computed as $x_i + T_g + T_{oa}$, which are all a round number of ticks by definition. T_{oa} is the precomputed 'time on air', in whole ticks. As noted before [35], usually $T_a \neq T_{oa}$. Now the estimated time difference can be rewritten into:

$$\begin{aligned}\delta_q x_{ij} &= q(x_j + T_g + T_a) - (x_i + T_g + T_{oa}) \\ &= q(\delta x_{ij} + x_i + T_a) - (x_i + T_{oa}) \\ &= q(\delta x_{ij} + T_a - T_{oa}) \end{aligned} \tag{5-3}$$

We first cancel out the guard times (which are always round numbers), then substitute $\delta x_{ij} + x_i$ for x_j, which allows us to cancel out x_i. This yields the final quantization rule, showing that the misestimation of the phase difference is influenced by the quantization operator itself and the difference between the (precomputed) T_{oa} and the real T_a.

In an unfortunate case such as displayed in Figure 5-2, where $\delta x_{ij} = 2.2, T_a = 5.1, T_{oa} = 6$ (all in ticks) the perceived time difference is $\delta_q x_{ij} = 7 - 6 = 1$, which is wrong by more than a tick. This is not unrealistic: the computation of T_{oa} is more accurate but is cast to an integer at the end, which amounts to a round down operation.

Since this quantization effect severely complicates most attempts at modelling the synchronization, it will initially be ignored. A common approach to quantization operators is to treat them as a disturbance, frequently taking the form of zero-mean additive noise [19]. It is clear however that the quantization in this case cannot be cast aside so easily since it contains a fixed term (the deterministic part) and is very dependent on the value of δx_{ij} itself. In section 5-5 we will briefly revisit quantization.

5-1-5 Objective

As was noted in chapter 2, synchronization is an elusive objective. In the strongest sense, synchronization is reached when all clocks achieve perfect timing with respect to some absolute time:

$$\tau_1(t) = \tau_2(t) = \ldots = \tau_N(t) = t$$

In the typical network setting of MyriaNed, an external or real time t is not available, so the strongest desired synchronization must be somewhat relaxed:

$$\tau_1(t) = \tau_2(t) = \ldots = \tau_N(t) \approx t$$

In addition, if nodes are not neighbours their clocks being out of sync is no trouble for gMAC. The only measure that influences the eventual energy consumption is the (maximum) absolute difference between the phases of neighbours. Perfect synchronization for a single round in this context can be defined as:

$$\begin{aligned} &\text{In a round } k: \\ &x_j(k) - x_i(k) = 0 \quad \forall (i,j) \in \mathcal{E}(k) \end{aligned} \tag{5-4}$$

Achieving this perfect synchronization is impossible, because it would require instantaneous compensation of the unknown and partially stochastic clock variation d. The performance of a synchronization algorithm in the context of MyriaNed is measured by the minimal maximum phase difference over all rounds and connections: this defines appropriate guard times. The trivial solution, where there are no links and thus no errors, is not allowed.

In the case of connected networks (networks for which there is a chain of links between any two nodes in the network), the objective stated in Equation 5-4 implies $x_1(k) = x_2(k) =$

$\ldots = x_N(k)$. This problem is in fact a consensus problem: the problem of having all nodes in a network agree on a common value.

Note that this is a slightly stronger problem statement than we initially had. To solve our synchronization problem, it is sufficient to keep the difference between neighboring nodes small. In consensus, we want all nodes to have an identical value. The problem of keeping differences between neighbouring nodes small has been named a gradient problem for obvious reasons [18, 64], but that term does not seem to have caught on. In the next section, we will detail the connection to consensus.

5-2 Consensus

In [49] a good survey of consensus processes is given. We will reproduce the most relevant theory here to familiarize the reader with basic consensus. The most straightforward and classic node-level algorithm of solving the consensus problem is:

$$\dot{x}_i(t) = \sum_{j \in \mathcal{N}_i} a_{ij}(x_j(t) - x_i(t)), \quad x_i(0) = b_i \tag{5-5}$$

where $x(t)$ is a continuous-time state of the node, which is more general than the phase. The weighting $a_{ij} \in [0, 1]$ is related to the connections w_{ij} previously mentioned by the fact that when w_{ij} is 0 (no connection), a_{ij} is necessarily 0 too. The a_{ij} terms can be used for tuning of the algorithm however, with the limitation that $\sum_j a_{ij} = 1$. In the coming descriptions of algorithms, we will reuse a_{ij} for algorithm-defined weights of links, but its role may change subtly between algorithms.

In the continuous-time form, the system's consensus process can be conveniently written as:

$$\dot{x} = -Lx \tag{5-6}$$

$$L = [l_{ij}] \tag{5-7}$$

$$l_{ij} = \begin{cases} -a_{ij}, & j \in \mathcal{N}_i \\ \sum_{j \neq i} a_{ij}, & j = i \end{cases} \tag{5-8}$$

L is the graph Laplacian, which is characteristic of the graph and has many interesting spectral features. As long as the directed graph is strongly connected – that is, there is a path from any node to any other node in the network – this consensus process converges. This is easily proven via analysis of the spectrum of L. If the graph is undirected or balanced (every node has as many incoming as outgoing links), the equilibrium value is the average of the initial states.

More relevant to our case is the canonical discrete-time consensus process:

$$x_i(k+1) = x_i(k) + k_p \sum_{j \in \mathcal{N}_i} a_{ij}(x_j(k) - x_i(k)) \tag{5-9}$$

where the similarity to Equation 5-2 is obvious. The factor $k_p > 0$ is a gain, and $a_{ij} \in [0, 1]$ are weights that can be defined in the algorithm. The collective dynamics under this rule are:

$$x(k+1) = Px(k)$$
$$P = I - k_p L$$

where P is commonly called the Perron matrix. In [49], it is proven that as long as $0 < k_p < \Delta$, where Δ is the maximum degree ($\sum_{j \in \mathcal{N}_i} a_{ji}$) in the network, consensus is

reached by this rule on a strongly connected network. Again, if the graph is balanced or undirected, the equilibrium is the average of initial states.

The synchronization algorithm that is currently used by MyriaNed (Median, introduced in chapter 3) is closely related to this consensus algorithm, albeit on a changing network topology, using quantized values, with an operator that is not very common in control systems. By discretizing the synchronization process into rounds of communication, the discretized consensus model fits perfectly: the change between two time steps is what is specified by the controller. The influence of the network topology will be a topic in the following section.

First the correspondence between the median operator and a general consensus algorithm will be made more explicit. The median operator can be seen as a linear operator in two ways: treating it as a weighted sum, or treating it as an approximate average.

Median as Weighted Sum The Median control algorithm takes the median of the incoming set of phase differences and corrects its own phase with half of that value:

$$\epsilon_i(k) = \frac{1}{2}\,\mathrm{Med}(\theta_i(k)) \tag{5-10}$$

where the Med function is defined as follows. Let $z \in \mathbb{R}^n$ be a sorted column vector, where $z_1 \leq z_2 \leq \ldots \leq z_n$.

$$\mathrm{Med} : \mathbb{R}^n \to \mathbb{R}, \text{such that } \mathrm{Med}(z) = \begin{cases} z_{\frac{n}{2}}, & n \text{ even} \\ z_{\frac{n+1}{2}}, & n \text{ odd} \\ 0, & n = 0 \text{ (empty vector)} \end{cases}$$

In essence, this is a selection. We can rewrite Equation 5-10 as a weighted sum:

$$\begin{aligned} \epsilon_i(k) &= \frac{1}{2}\,\mathrm{Med}(\theta_i(k)) \\ &= \frac{1}{2} \sum_{j \in \mathcal{N}_i(k)} a_{ij}(k)(x_j(k) - x_i(k)) \\ a_{ij}(k) &= \begin{cases} 1, & \text{if } j \text{ is the median node of } i \\ 0, & \text{else} \end{cases} \end{aligned}$$

Now the state update equation per node is:

$$x_i(k+1) = x_i(k) + d_i + \frac{1}{2} \sum_{j \in \mathcal{N}_i(k)} a_{ij}(k)(x_j(k) - x_i(k)), \quad x(0) = b$$

which is the general discrete-time consensus process from Equation 5-9 with time-varying links, weights and a constant added per round, and $k_p = \frac{1}{2}$. Integrating this into the general model of Equation 5-2 yields:

$$\begin{aligned} &x(k+1) = A_m(k)x(k) + d \qquad (5\text{-}11) \\ &\text{where:} \\ &A_m(k) = [a_{ij}(k)] \\ &a_{ij}(k) = \begin{cases} 1 - \sum_{j \neq i} a_{ij}(k) = 0.5, & i = j \\ 0.5, & \text{if } j \text{ is the median neighbour of } i \\ 0, & \text{else} \end{cases} \end{aligned}$$

Here we are using the subscript m to median selection, as opposed to average.

Median as Approximate Average Instead of a selection, one could also choose the a_{ij} weights in Equation 5-9 so as to compute the average difference with neighbours and correct the state with that. This is a more common approach to the consensus problem [57, 32]. To compensate the difference between the average and the median, the extra term $v(k)$ is introduced.

$$\begin{aligned} x(k+1) &= A_a(k)x(k) + v(k) + d \\ &\text{where:} \\ A_a(k) &= [a_{ij}(k)] \\ a_{ij}(k) &= \begin{cases} 1 - \sum_{j \neq i} a_{ij}(k), & i = j \\ \frac{1}{|\mathcal{N}_i|}, & \text{if } j \in \mathcal{N}_i \\ 0, & \text{else} \end{cases} \end{aligned} \tag{5-12}$$

This $v(k)$ is a stochastic vector that is zero-mean, as proven by the following lemma:

Lemma 5-2.1. *If the underlying distribution is symmetric and has a mean, the sample median is an unbiased approximator for the sample mean.*

Proof. Let $X_1, \ldots, X_n$ be the sample of a symmetric distribution with mean μ. Let $Y_i = X_i - \mu$ be samples corrected for that mean. Now take $\hat{\mu} = \mathbb{E}(\text{ Med}(Y_1, \ldots, Y_n))$. By symmetry of the samples Y_i around 0, $-\hat{\mu} = \mathbb{E}(-\text{ Med}(Y_1, \ldots, Y_n)) = \mathbb{E}(\text{ Med}(Y_1, \ldots, Y_n))$. Since $-\hat{\mu} = \hat{\mu}$, it must be 0. Now

$$\begin{aligned} \mathbb{E}(\text{ Med}(X_1, \ldots, X_n)) &= \mathbb{E}(\text{ Med}(Y_1 + \mu, \ldots, Y_n + \mu)) \\ &= \mathbb{E}(\mu + \text{ Med}(Y_1, \ldots, Y_n))) \\ &= \mu \end{aligned}$$

□

The symmetry assumption is satisfied by the phase differences in the network, since the process that generates these differences is perfectly symmetric. For any increased probability of a positive time difference between node i and j, an equal increase in probability of a negative time difference between node j and i arises. This guarantees that the underlying global distribution of time differences is symmetric. This means that $v(k)$ in Equation 5-12 is zero-mean.

Consensus algorithms of this form, where the controller acts as a single integrator with some modified input, are called first order consensus algorithms. In the next section, ways of proving stability of such algorithms on discrete, switching, directed networks will be introduced. Improved synchronization algorithms will involve drift estimation (see chapter 4), which can not be expressed by first order consensus algorithms: a second state (and thus a second order) is needed. A general description of second-order linear consensus algorithms is:

$$x_i(k+1) = x_i(k) + k_p \sum_{j \in \mathcal{N}_i} a_{ij}(x_j(k) - x_i(k)) + k_i \alpha_i(k) + d_i \tag{5-13}$$

$$\alpha_i(k+1) = \alpha_i(k) + f(\theta_i(k)) \tag{5-14}$$

Where f is a linear function, and k_i a tuning factor. Analysis of stability of these algorithms proves to be very difficult, and some attempts will be made in section 5-4.

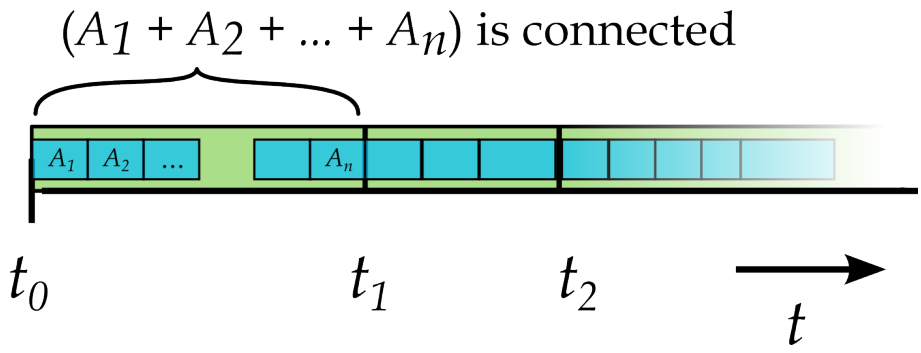


Figure 5-3: A visualization of the condition for convergence of discrete consensus on switching *undirected* graphs.

5-3 First Order Algorithms

This section will provide an overview of the techniques available to prove stability and convergence of consensus algorithms as described by Equation 5-9, with an application to synchronization protocols in MyriaNed. Proving convergence under ideal circumstances can be done using several techniques from literature. When we consider a disturbed consensus process on a switching, directed network however, most techniques are of limited value.

Matrix Theory

In [32], one of the earlier publications on consensus, a discrete time consensus process on switching undirected networks is treated. Using the ergodicity of the system matrix and its products, Wolfowitz's Lemma (see Appendix A-1-2) is invoked to prove convergence towards a consensus value. Using this technique, it can be concluded that (paraphrasing):

- If every (undirected) topology that occurs is connected, the system converges to consensus.
- If there is an infinite sequence of contiguous, nonempty, bounded, time-intervals $[t_i, t_{i+1})$, starting at t_0 across each of which the union of topologies is connected, the system converges. This is visualized in Figure 5-3.

Especially the second conclusion, which implies the first one, is very interesting. If we would accumulate all the links that occur over the rounds of gMAC, and after some time these links form a connected network and we could repeat this process again, consensus will converge. Since the links in MyriaNed are probabilistic, this will be the case as long as all nodes are close enough to form one network. It is very unlikely that one round will contain all links to create a connected network, but it is bound to happen over multiple rounds. As long as the links that are needed to make the network connected have some probability of happening, it will happen eventually and the condition is satisfied.

MyriaNed is a directed network however. The first conclusion can be extended to the directed case by using the Geršgorin Circle Theorem (see Appendix A-1-1) to limit eigenvalue locations, but this does not suffice for extending the second, more useful result. This is generalized to directed networks with arbitrary varying positive weights by [53]. This framework comes closest to proving the stability of the Median algorithm in the case of perfect, identical oscillators. Consider the system:

$$x_i(k+1) = \frac{1}{\sum_{j=1}^{n} a_{ij}(k)G_{ij}(k)} \sum_{j=1}^{n} a_{ij}(k)G_{ij}(k)x_j(k)$$

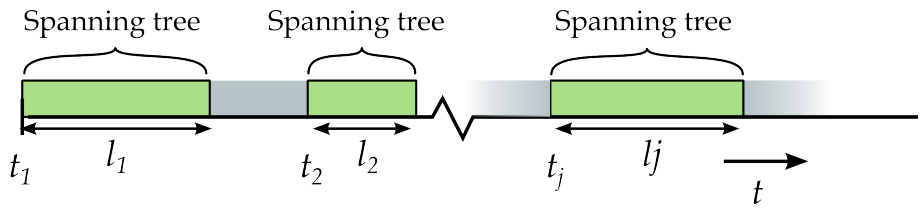


Figure 5-4: A visualization of the condition for convergence of discrete consensus on switching *directed* graphs.

Where G_{ij} is 1 or 0 according to the existence of link (i, j) ($G_{ii}\hat{=}1$), and $a_{ij} > 0$ are weights given to certain neighbours. Within this model we are still free to apply the 'median as average' an 'median as selection' interpretations. In matrix form:

$$\begin{aligned} x_i(k+1) &= D(k)x_i(k) \\ D &= [d_{ij}] \\ d_{ij} &= a_{ij}(k)G_{ij}(k) / \sum_{j=1}^{n} a_{ij}(k)G_{ij}(k) \end{aligned} \tag{5-15}$$

The main theorem of [53] is as follows (paraphrased): The system (5-15) achieves consensus asymptotically over a sequence of topologies $\mathcal{G}(k)$ if there exists an infinite sequence of uniformly bounded (see Definition A.5), nonoverlapping time intervals $[t_j, (t_j + l_j)), j = 1, 2, \ldots$, starting at $t_1 = 0$, with the property that each interval $[t_j + l_j, t_{j+1})$ is uniformly bounded and the union of graphs across each interval $[t_j, (t_j + l_j))$ has a spanning tree. This condition is visualized in Figure 5-4.

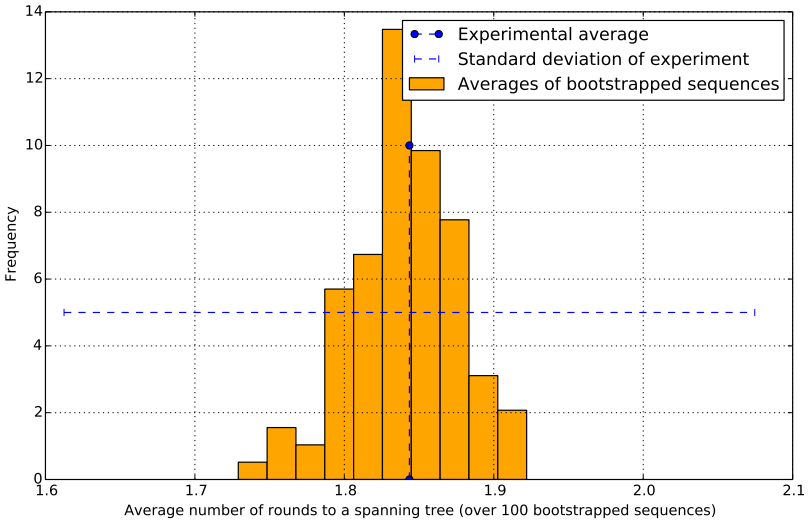
In other words: all the nodes in the network will agree on a common value (not necessarily the average) as long as the network topology forms a spanning tree over intervals that occur often enough, are short enough and are separated by intervals that are not too long. A network is said to contain a spanning tree if every node in the network except for one (the root node) has at least one incoming connection, and the network is connected. A strongly connected network has a spanning tree.

The consensus rule is not exactly equal to either (5-12) or (5-11). What matters for the proofs is that the system matrices are all stochastic, nonnegative, square and positive on the diagonal. In practice this means that the weights a_{ij} used in an algorithm must be between 0 and 1, and the sum $\sum_{j \neq i} a_{ij} < 1$, in order to keep the diagonal entry positive. The proof is then produced by first establishing that if a sum of matrices (union of graphs) contains a spanning tree, the product of matrices will contain a spanning tree as well. Secondly, matrices describing graphs with a spanning tree are shown to have a single maximum eigenvalue at 1. This is quite intuitive, since in order to create a spanning tree, at least $N - 1$ nodes will need to have at least one incoming link, so there will be only one row at most with all zero off-diagonal entries. Via Geršgorin, the single maximum eigenvalue at 1 follows naturally. Finally, stochastic matrices with this spectral property are ergodic, and then via an ordering of the union intervals and Wolfowitz again, convergence is proven.

If looking at *all* transmissions, spanning trees occur very regularly in MyriaNed deployments. In a network where all nodes are within each others range, a spanning tree will occur as soon as one node broadcasts successfully to all others, which is bound to happen very quickly. To proceed with the above analysis, we would need to consider only the median neighbours as connections, and look at the occurrence of spanning trees. Since the variety of networks and possible phase differences is very large, it is impossible to reasonably estimate the likelihood of spanning trees analytically.

This analysis was done using experimental data from the two-group experiment from [35]. The topological information and measured time differences from this experiment were preserved over 300 rounds. This data was parsed, starting at the first round. 'Median connections' from rounds were added to the topology until a spanning tree is formed. When this is the case, rounds are removed from the start of this interval until the spanning tree disappears. The lengths of the intervals were averaged, showing in the blue line in Figure 5-5b, with the standard deviation. To make sure that this is not a lucky result, the topology information per round was bootstrapped: new sequences of topologies were generated by randomly picking rounds (with replacement) from the existing dataset. Doing this 100 times, and calculating the average every time shows that the result is not statistically surprising: spanning trees always occur frequently.

(a) Topology over the whole experiment



(b) The average number of rounds to a spanning tree, and its likelihood

Figure 5-5: Results from a two-group experiment carried out earlier. The data was analyzed to get an impression of the frequency of occurrence of spanning trees.

Lyapunov Theory

A second promising approach is Lyapunov theory: attempting to construct a Lyapunov function for the global system, which proves stability (see Appendix A-1-3). A continuous time consensus process is studied in [48]. The network under study is a directed, weighted, switching topology, and follows the process described by Equation 5-5. This leads to a network dynamic model such as Equation 5-8, but with a time-variable Laplacian matrix. The graph is assumed to be strongly connected and balanced. There are several limitations to this model that keep it far from describing MyriaNed: it has a continuous-time state,

every instance of the graph has to be strongly connected in itself, and it needs to be balanced.

The balanced property is needed because it makes the Laplacian positive semi-definite, allowing the following Lyapunov approach. A disagreement vector $\delta(t) = x(t) - x_{ss}$ is defined, where x_{ss} is the consensus value, which happens to be the average of $x(0)$ under the above assumptions. It has dynamics:

$$\dot{\delta}(t) = -\mathcal{L}(k)\delta(t)$$

It is then proven that a valid Lyapunov function is

$$V(\delta) = \frac{1}{2}\|\delta\|^2$$

which is smooth, positive-definite, so asymptotic stability with a performance bound is proven.

In [43] the Lyapunov approach is taken too, but on a level of extreme abstraction. A very general nonlinear discrete-time system on an Euclidian space is investigated:

$$x(k+1) = f(k, x(k))$$

A sequence of directed graphs $(\mathcal{N}, \mathcal{A}(k))$ (the node set and adjacency matrix) determines what information is available to each node at a specific time. If f is such that for every node the updated state $x_i(k+1)$ is a strictly convex combination of its current state $x_i(k)$ and the current states of its neighbours, the convex hull of states $\tilde{V}(x) = \text{conv}\{x_1, ..., x_n\}$ does not grow along the solutions of the system: $\tilde{V}(f(t, x)) \leq \tilde{V}(x)$. Thus, the system is (Lyapunov) stable.

Convergence towards the equilibrium requires conditions on the topology. The system is uniformly globally attractive if and only if there is an interval $T \geq 0$ such that for all time instances t_0 there is a node connected to all other nodes across the interval $[t_0, t_0 + T]$. In essence, this is a similar requirement as the spanning tree requirement from [53]. A node being connected to another node across an interval does not place demands on the order of links between these nodes. It is again just the union of the graphs that occur in the interval that matters.

Note that x_k does not have to be a scalar per node in this case, and certain second-order processes can be analyzed as well. We will attempt this in the following section.

The median operation for node i can be simplified to:

$$x_i(k+1) = \frac{1}{2}x_i(k) + \frac{1}{2}x_j(k)$$

Where j is the index of the median for this round. This is always strictly within the convex hull of the states of i and its neighbours (see appendix A-2). Only when a node has no neighbours, this rule is violated.

In conclusion, we have been able to establish two subtly different results via the two approaches:

1. The Median algorithm is convergent if just the connections formed by the median selection operation form a spanning tree in the network accross bounded time intervals that form an infinite sequence. It is even allowed that there are bounded intervals in between the spanning-tree-containing intervals. This result is valid for all algorithms where the new state of a node is a weighted sum of its own and its neighbours' states, and the weights sum up to 1.

2. The Median algorithm is convergent if the network forms a spanning tree within a bounded interval starting at any time. This result is valid for any process where a node's new state is a convex combination of its own state and that of its neighbours.

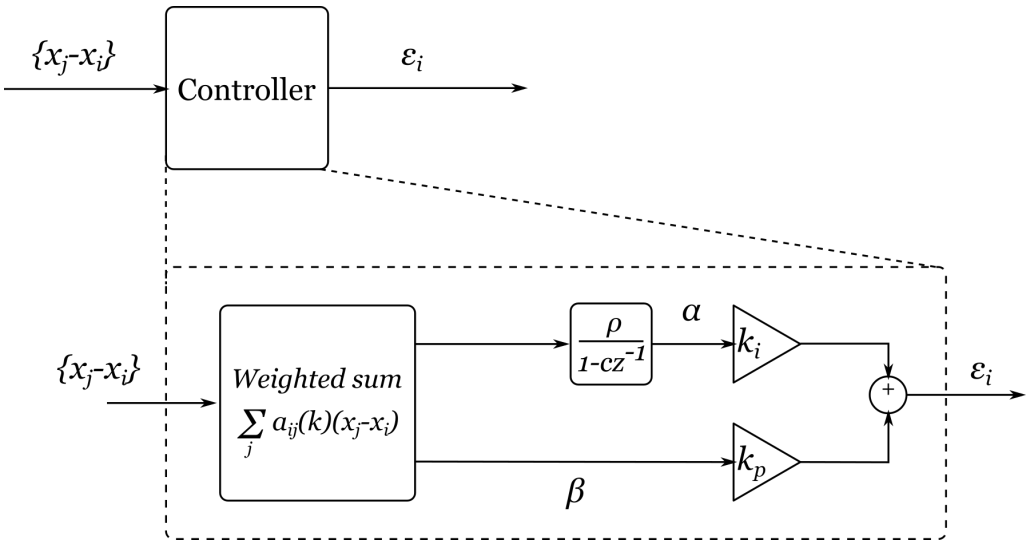


Figure 5-6: The controller formulated in a way that encompasses Median, MemoryMedian and PISync.

5-4 Second Order Algorithms

Out of chapter 4 we concluded that drift estimation is key to improving synchronization performance. As we will see in chapter 6 and chapter 7, two drift-estimating algorithms emerge as serious contenders for an improvement over Median: MemoryMedian and PISync. The algorithms are very similar, and we will first try to formulate them in the same framework in order to analyze their properties and differences.

5-4-1 Framework

The controller structure for Median, MemoryMedian and PISync as it would fit into Figure 5-1 is shown in Figure 5-6. The set of measured time differences is first accumulated in a weighted sum (for PISync, two different weighted sums). Unfortunately, these sums are not time-invariant but different per round number k. The resulting values are used for proportional correction (β multiplied by k_p) to ensure a good enough synchronization in the next round. Additionally, another value weighted sum is fed through a low pass filter (integrator) to contribute to the long-term error estimate. This estimate α is used for correction via k_i. A special case is when no time differences are received, then only the latest value from the low pass filter is used for correction.

5-4-2 Median

For Median, the diagram in Figure 5-6 simplifies a good deal. The weighted sum becomes the median operation (thus one a_{ij} coefficient is 1, the others 0), and $k_i = 0$, cancelling the drift estimation branch. k_p is commonly 0.5.

5-4-3 MemoryMedian

In Figure 5-7, the extension of Median to MemoryMedian is shown. The weighted sum has become a median operation. The filter that results in α can be used in two ways, $c = 1$ and $c = 1 - \rho$. For $c = 1$, the time-domain filter equation is $\alpha_i(k+1) = \alpha_i(k) + \rho \ \mathrm{Med}(\theta_i(k))$.

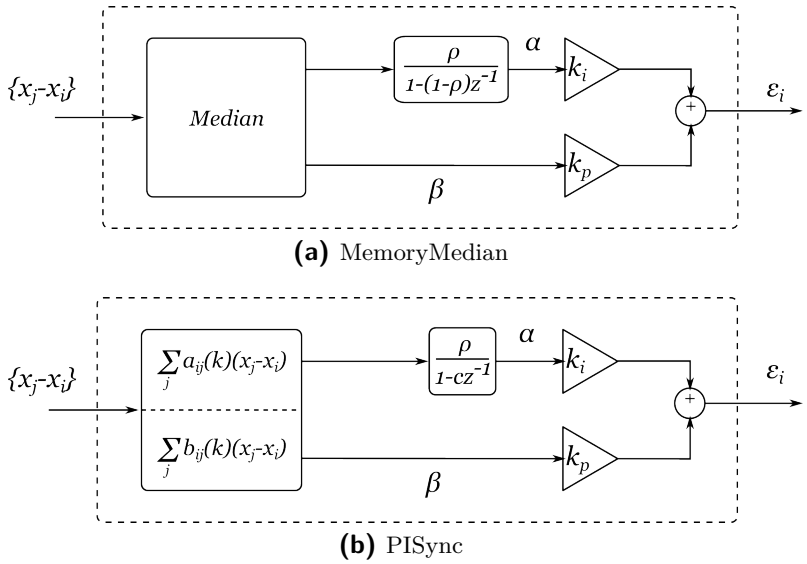


Figure 5-7: The general controller from Figure 5-6, specified into MemoryMedian and PISync.

For $c = 1-\rho$, it is s $\alpha_i(k+1) = (1-\rho)\alpha_i(k) + \rho\ \mathrm{Med}(\theta_i(k))$. $\alpha_i(k)$ is the state and output of the filter; the long-term estimate of the drift or the correction.

The first implementation ($c = 1$) is most truthful: the error gets integrated with a certain gain, and as long as the measured error stays zero, this integrated value stays constant. This filter is susceptible to windup (any small error gets added) and has the drawback that its granularity is determined by ρ. The granularity determines the accuracy of the output and the convergence speed. We might want to pick a large value for ρ to make the filter adjust quickly to changing errors, but this implies a low precision in the values that α_i can take.

The other option is setting $c = 1 - \rho$. This filter does not have the coupled convergence and granularity problem of the other implementation because a large ρ means $1 - \rho$ is small and vice versa. Since this filter needs a persistent input to keep α_i at a constant level, it does not wind up easily. It has a drawback as well: since it needs an error to keep its level, there will always be a residual error while using this filter. A measured error of 0 will change the value of α_i, and in the next round this will create a new time difference. In simulations, this filter turned out to perform best however.

5-4-4 PISync

For PISync, the block diagram is specified in Figure 5-7b. The weighted sum block has become a bit more complicated: there is a different set of weights for the proportional and integral correction. In the algorithm as it is ported to MyriaNed, the time differences are first multiplied by a number as they come in, and then the average is taken. Of course this can be described in one step by a weighted sum as well.

For the proportional correction a constant factor is used, b_c, so $b_{ij}(k) = \frac{b_c}{N_j(k)}$, where $N_j(k)$ is the number of neighbours in round k. The factor k_p is now redundant, and is set to 1.

For the integrating filter, the same reasoning as above holds. The gains in the weighted sum are extremely small however, so even with $c = 1$ and $\rho = 1$ the granularity of the estimate is fine enough, because the input is a very small number. It was noted in simulations that the integration of the transmission time estimate error still leads to integrator windup, and thus an overall speedup of the network. A filter structure with $\rho = 1$ and $c < 1$ will prove most useful, see section 7-3-4.

In the following section we will try to apply techniques from first order algorithms to these second order algorithms.

5-4-5 Stability

The system matrix approach is difficult to apply to these algorithms. In addition to the single state per node (the phase error), the filters add another local state. The system per node is now:

$$x_i(k+1) = x_i(k) + d_i + k_p \sum_{j \in \mathcal{N}_i(k)} b_{ij}(k)(x_j(k) - x_i(k)) + k_i \alpha_i(k)$$
$$\alpha_i(k+1) = c\alpha_i(k) + \rho \sum_{j \in \mathcal{N}_i(k)} a_{ij}(k)(x_j(k) - x_i(k))$$

The state α_i has the purpose of compensating the persistent disturbance (d_i), and thus it is not required that it should go to 0 or be in agreement with all other α's. This is a very different objective than defined in the proportional algorithms. Even if solve this problem by doing a state transformation $\gamma_i(k) = d_i + k_i\alpha_i(k)$ such that the new state γ_i needs to approach 0, the resulting system matrix will have row sums of more than zero, since x is a consensus process plus some other process. This violates a basic assumption in all matrix approaches: that the system matrix is stochastic.

The Lyapunov approach seems more promising, since it allows for multiple states naturally. Unfortunately, the convexity constraint is violated: there is no process that ensures $\alpha_i(k+1)$ is in the convex hull of $\alpha_i(k)$ and the α's of its neighbours. Although commonly $c < 1$, all neighbours might have an α that is farther removed from 0, and in that situation the convex hull might become larger (if all phase differences are 0 for example). If $c = 1$ (or larger), the strict convexity is violated as well, if $\alpha_i(k)$ happens to be on the convex hull.

5-5 Error Bounds

In the analysis of first order algorithms and their stability we have ignored the effect of drift. Since we conceive of this drift as a disturbance instead of a characteristic of the plant, this is allowed, and Median could be proven to converge. When adding the disturbance, this will drive the states of nodes away from perfect consensus. In such a process, an equilibrium will be found between the convergence of the controlling algorithm and the divergence of the disturbance. Finding the position of this equilibrium would be extremely useful, because it would allow us to analytically predict the guard times that a network needs.

The diverging properties of the disturbance are easily modelled using assumptions on the drift values and round time. Providing bounds on the convergence of consensus processes is much more involved. The dynamics of the topology are quickly complicating matters for networks other than fully connected, undirected networks. No results have been published for all but the most simple networks [34, 5], and bounds are often found to be overly conservative when compared to numerical results [23].

Chapter 6

Candidates

In chapter 4, multiple promising algorithms that are compatible with gossip MAC (gMAC) were found. In this chapter, their MyriaNed-specific versions will be introduced. Every algorithm will first be repeated as it was designed originally. Then the steps for adaptation will be described, and a pseudocode version will be provided. The notation will be the same as in 5-1

6-1 MemoryMedian

6-1-1 Median

The original Median algorithm does only proportional correction, based on the Median of time differences. We have already introduced the algorithm in words (chapter 3) and in a block diagram (section 5-4), but to be consistent with the other sections of this chapter, it is reformulated in pseudocode in Algorithm 1.

Algorithm 1 Median, as executed by a node i

```
for every round k do                                  ▷ After receiving messages
    for every received message j do
        Infer phase difference (x_j − x_i)
        Add it to θ_i(k)
    end for
    Compute ε_i(k) ← k_p Med(θ_i(k))
    T_{i,k} ← T + ε_i(k)                              ▷ Correct the coming idle time
end for
```

6-1-2 MemoryMedian

As concluded in chapter 4, the inclusion of drift estimation would likely be an improvement. We adhere to the heuristic that the median of phase differences is a good representative of the group of phase differences.

By integrating the median values, the drift can be compensated in the long term. Two filters were considered:

Table 6-1: MemoryMedian properties

	Parameter	Range	Use
Message	None		
Tuning parameters	ρ	$(0, 1)$	Lowpass of time difference integration
	k_p	$(0, 1)$	Gain of proportional correction
	k_i	$(0, 1)$	Gain of integral correction

1. a cumulative one:

$$\alpha_i(k+1) = \alpha_i(k) + \rho \, \text{Med}(\theta_i(k))$$

2. a balanced one:

$$\alpha_i(k+1) = (1-\rho)\alpha_i(k) + \rho \, \text{Med}(\theta_i(k))$$

In the simulations that will be treated in chapter 7, these two filters are compared. Theoretically, the cumulative filter works better. It keeps integrating errors until a perfect estimate is reached, and then keeps its estimate as long as the error is zero. The balanced filter on the other hand needs a constant error input to retain its estimate, and this a steady-state error is unavoidable. It turns out however that even in a small, unchallenging topology the cumulative filter suffers from windup. The second balanced filter is a more robust alternative, and is therefore preferred.

The MemoryMedian algorithm is described in Algorithm 2. For $k_p = 0.5$ and $k_i = 0$, this equals Median.

Algorithm 2 MemoryMedian, as executed by a node i

```
for every round k do                                  ▷ After receiving messages
    for every received message j do
        Infer phase difference (x_j − x_i)
        Add it to θ_i(k)
    end for
    β_i(k) ← Med(θ_i(k))
    α_i(k) ← (1 − ρ)α_i(k − 1) + ρβ_i(k)              ▷ Integrate the time difference
    ε_i(k) ← k_i α_i(k) + k_p β_i(k)
    T_{i,k} ← T + ε_i(k)                              ▷ Correct the coming idle time
end for
```

6-2 PISync

In [9, 75], a PI controller for time synchronization is proposed, specially designed for a network with asynchronous, directional gossip communications. The same authors had published some previous iterations of this approach, but the algorithm described in [9] is reportedly most useful for real-life deployments.

6-2-1 Original

A linear clock model is again the starting point. The offset is not an initial quantity, but instead the time progression since the last update is used. Let $t_{u,i}$ denote the time instant

update u on node i. In addition let $t'_{u,i}$ be the time instant just before the update is applied. Then the clock of node i is

$$\tau_i(t) = \tau_i(t_{u,i}) + \hat{a}_i(t_{u,i})[t - t_{u,i}]$$

Where $\hat{a}_i$ is a drift estimate that is used to compensate the real drift a_i. In the next update instant both quantities characterizing the clock readout are updated:

$$\tau_i(t_{u,i}) \leftarrow \tau_i(t'_{u,i}) + u'(t_{u,i})$$
$$\hat{a}_i(t_{u,i}) \leftarrow \hat{a}_i(t'_{u,i}) + u''(t_{u,i})$$

Both inputs u are conveniently already functions of the time difference with a neighbouring node:

$$u'(t_{u,i}) = b_{ij}(\tau_j(t'_{u,i}) - \tau_i(t'_{u,i}))$$
$$u''(t_{u,i}) = a_{ij}(\tau_j(t'_{u,i}) - \tau_i(t'_{u,i}))$$

The values of a_{ij} and b_{ij} vary across publications. In [9] $b_{ij} = 0.5$ is not even considered tunable. For fixed values of a_{ij}, b_{ij}, [8] proves convergence of the algorithm when $b_{ij} \in (0, 1]$, and $a_{ij} \in (0, \bar{a}_{ij})$, where $\bar{a}_{ij}$ depends on some properties of the graph Laplacian. In [75] an adaptive scheme is proposed, where the value of a_{ij} is dependent on the differential of the measured error:

$$a_{ij}(k) = \begin{cases} 0 & \text{if } |\tau_j(k) - \tau_i(k)| > e_{max} \\ a_{max} & \text{if } |\tau_j(k-1) - \tau_i(k-1)| > e_{max} \text{ and } |\tau_j(k) - \tau_i(k)| \leq e_{max} \\ \lambda_i(k) a_{ij}(k-1) & \text{otherwise} \end{cases}$$

$$\lambda_j(k) = \begin{cases} 1 & \text{if } \tau_j(k-1) - \tau_i(k-1) = 0 \text{ or} \\ & \tau_j(k) - \tau_i(k) = \tau_j(k-1) - \tau_i(k-1) \\ \min\left(\left|\frac{\tau_j(k-1) - \tau_i(k-1)}{(\tau_j(k) - \tau_i(k)) - (\tau_j(k-1) - \tau_i(k-1))}\right|, \frac{a_{max}}{a_i(k-1)}\right) & \text{otherwise} \end{cases} \tag{6-1}$$

Foreshadowing the MyriaNed implementation, the update instants $t_{u,i}$ are represented by k here for ease of notation.

The logic behind this approach is as follows. To prevent integrator windup, errors that can not reasonably stem from drift accumulation should not be integrated. This leads to a choice for e_{max}. This situation is expected at the start of a synchronization, when offset is the main contribution to time differences. Then as soon as the offset is diminished sufficiently, and the e_{max}-boundary is traversed, the maximum integral gain a_{max} should be used to promote fast correction. A large a_{ij} is thought to be causing a large steady state error due to noise integration. Thus, when closer to a noise-disturbed steady state, a_{ij} should be smaller. If there is a time difference caused by offsets, the error is typically quite constant, and the magnitude is large compared to the difference over rounds. In (6-1), this is translated into suitable values for λ, where division-by-zero anomalies and exceeding of a_{max} are prevented.

6-2-2 Adaptation

The communication strategy of MyriaNed fits this protocol precisely, and there is no need for extra communication overhead. One discrepancy is however that the original PI controller updates after every received message, whereas MyriaNed can only update after a series of messages has been retrieved. Updating between different messages would

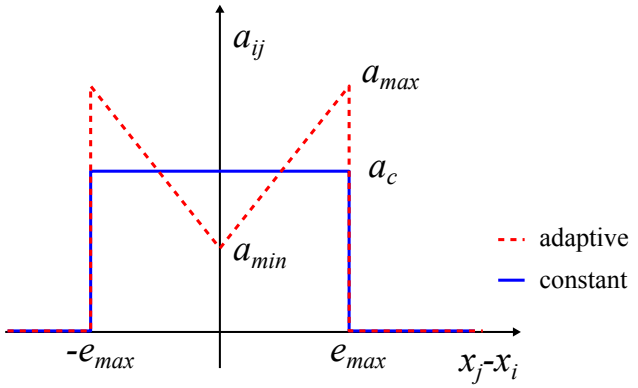


Figure 6-1: Different strategies for determining a_{ij} for PISync

inevitably lead to timing issues: the processing of a message might not finish in time, and negative corrections are very limited. We will solve this issue as shown in Algorithm 3: messages are collected over a round, and the corrections are applied all at once.

This makes the adaptive gains hard to replicate precisely. The rules are designed by the authors for the case when every received message is followed by a correction, not a 'batch' correction. As a compromise, we propose two schemes: a constant a_c, which is only applied for $|\tau_j(k) - \tau_i(k)| \leq e_{max}$, and an 'adaptive' scheme, where a_{ij} is varied between a_{max} for $|\tau_j(k) - \tau_i(k)| = e_{max}$ and a_{min} for $\tau_j(k) - \tau_i(k) = 0$, see Figure 6-1. This scheme was proposed by the authors of [75] before, in an unpublished version of their paper.

Since the amount of correction done to the clock and drift estimate should not grow when more messages are received, the average correction that follows from the individual corrections is used.

In this algorithm α_i is not a direct estimate of the drift value, but an estimate of $\frac{1}{f_i}$, the inverse of a node's frequency. To preserve units and gain magnitudes, this convention is kept. The update equation looks slightly different. Let $\tilde{\alpha}_i$ be an estimate of the node's drift, then $\frac{1}{\tilde{\alpha}_i} = \frac{f_n}{f_i} = f_n\alpha_i$, f_n being the nominal frequency.

Algorithm 3 PISync for MyriaNed, as executed by a node i

$\alpha_i = \frac{1}{f_n}$ ▷ The initial estimate is 1/32kHz
for every round k **do** ▷ After receiving messages
  for every received message j **do**
    Infer phase difference $\delta x_{ij} = x_j - x_i$
    Compute $a_{ij}(\delta x_{ij})$ ▷ The (optionally) adaptive value of a
    Record $b\delta x_{ij}$ ▷ Aggregate u'
    Record $a_{ij}\delta x_{ij}$ ▷ Aggregate u''
  end for
  $\beta_i(k) \leftarrow$ Mean($\{b\delta x_{ij}\}$)
  $\alpha_i(k) \leftarrow \alpha_i(k-1) +$ Mean($\{a_{ij}\delta x_{ij}\}$) ▷ Update drift estimate
  $\epsilon_i(k) \leftarrow (f_n\alpha_i(k) - 1)T + \beta_i(k)$
  $T_{i,k} \leftarrow T + \epsilon_i(k)$ ▷ Correct the coming idle time
end for

6-3 ATS: Average TimeSync

The most different algorithm that still fits MyriaNed is ATS [57], which relies on two simultaneous consensus processes to have all nodes agree on the time.

Table 6-2: PISync for MyriaNed, properties

	Parameter	Range	Use
Message	none		
Tuning parameters	a_c or a_{min}, a_{max}	$(0, T/2)$	Drift estimate update speed
	e_{max}	> 0 (function of T)	Maximum error to integrate
	b	$(0, 1]$	Proportional gain

6-3-1 Original

ATS is different from the other protocols because it does not permanently correct its clock, but continuously estimates the parameters needed to translate its hardware clock value, τ_i, to a time estimate via $t \approx \hat{\tau}_i = \hat{a}_i \tau_i + \hat{b}_i$. The protocol consists of three parts: drift estimation, drift compensation and offset compensation.

Drift estimation Every node keeps estimates of the relative drift between itself and its neighbours. The drift factor of the neighbouring node in terms of node i's clock is η_{ij}:

$$\begin{aligned}\tau_i(t) &= a_i t + b_i \\ \tau_j(t) &= a_j t + b_j = \frac{a_j}{a_i}(\tau_i(t) - b_i) + b_j \\ &= \frac{a_j}{a_i}\tau_i(t) + b_j - \frac{a_j}{a_i} b_i \\ &= \theta_{ij}\tau_i(t) + b_j - \eta_{ij} b_i\end{aligned}$$

In round k, node i receives a timestamped message of node j at time instant $t_{k,j}$, and records the time of arrival. Assuming negligible transmission delays, this leads to two values at node i (the perceptions of $t_{k,j}$ by i and j): $\tau_i(t_{k,j}), \tau_j(t_{k,j})$. Repeating this a round later, $\tau_i(t_{k+1,j})$ and $\tau_j(t_{k+1,j})$ are retrieved. This is already sufficient to calculate the drift between the two nodes. Due to (quantization) errors and time variability, the estimate is smoothed by a low pass filter with a tuning parameter $\rho_\eta \in (0, 1)$:

$$\eta_{ij} \leftarrow \rho_\eta \eta_{ij} + (1 - \rho_\eta)\frac{\tau_j(t_{k+1,j}) - \tau_j(t_{k,j})}{\tau_i(t_{k+1,j}) - \tau_i(t_{k,j})} \tag{6-2}$$

Drift compensation These drift estimates are used to apply a consensus protocol on the drift value of the network as a whole. Every node tries then to correct towards a reference drift $\bar{a}$. Neighbouring nodes communicate their own drift estimate $\hat{a}_j$, and node i uses those to update its own estimate (again smoothed):

$$\hat{a}_i \leftarrow \rho_v \hat{a}_i + (1 - \rho_v)\eta_{ij}\hat{a}_j$$

This is essentially a consensus process, and is proven to converge to a common estimate $\bar{a}$ along those lines, such that for every node asymptotically $\hat{a}_i a_i = \bar{a}$.

Offset compensation The offset compensation works very similarly, and is executed at the same time as the drift compensation. The difference between the two corrected times is used to converge towards a common offset $\bar{b}$.

$$\hat{b}_i \leftarrow \rho_o \hat{b}_i + (1 - \rho_o)\left((\hat{a}_i \tau_i(t_{k,j}) + \hat{b}_i)) - (\hat{a}_j \tau_j(t_{k,j}) + \hat{b}_j))\right) \tag{6-3}$$

Note that this requires the sending of $\hat{b}_j$ as well. This process makes every node's estimate converge such that $\hat{b}_i = \bar{b} - \hat{a}_i b_i$ asymptotically.

Table 6-3: ATS for MyriaNed, properties

	Parameter	Range	Use
Message	$\hat{a}$	$1 \pm \mathcal{O}(10^{-4})$	Estimate of node's inverse drift such that $\hat{a}_i a_i = \bar{a}$
	$\hat{b}$	$\mathbb{N}$	Estimate of node's inverse offset [ticks] such that $\hat{b}_i - \hat{a}_i b_i = \bar{b}$
	τ_i	$\mathbb{N}^+$	Current hardware clock value
Tuning parameters	ρ_η	$(0,1)$	Lowpass of relative drift estimate
	ρ_v	$(0,1)$	Lowpass of reference drift estimate
	ρ_o	$(0,1)$	Lowpass of offset estimate

6-3-2 Adaptations

The protocol as proposed above needs to send three items per round: the time stamp, drift estimate and offset estimate. In ATS, it's important that the τ timestamp sent is the uncorrected clock, whereas the $\hat{\tau}$ is the corrected clock value. In other algorithms, the most recent version of the clock is always used, so corrections can be applied at any time, and readouts are always correct. Here we need to keep track of two clocks, the hardware and the (corrected) software clock.

ATS can be executed as intended, using every received message for updating the estimates. At the end of each round, the best clock estimate $\hat{\tau}$ can be used to determine the next wakeup time. Since $\hat{\tau}$ is the best estimate of the reference time, we should wake up at

$$\begin{aligned} \hat{\tau}_i &= (k+1)T \\ \hat{a}_i \tau_i + \hat{b}_i &= (k+1)T \\ \tau_i &= \frac{(k+1)T - \hat{b}_i}{\hat{a}_i} \end{aligned}$$

Since the gap between the current wake up time and the next is dependent on the estimates up to now, it can not be expressed as a simple correction of T. Via τ_i, the wakeup time of node i can be expressed as $t_{k+1,i} = a_i \frac{(k+1)T - \hat{b}_i}{\hat{a}_i} + b_i$. The time difference that will be used as a performance measure for the synchronization is $\hat{\tau}_j - \hat{\tau}_i$, since if this difference is zero, both nodes have possession over the same clock, and are perfectly synchronized.

Algorithm 4 ATS for MyriaNed, as executed by a node i

for every round k **do** ▷ After receiving messages
  for every received message $j : (\tau_j, \hat{b}_j, \hat{a}_j)$ **do**
    if node j was heard before **then**
      $\eta_{ij} \leftarrow \rho_\eta \eta_{ij} + (1 - \rho_\eta) \frac{\tau_j - \tau'_j}{\tau_i - \tau'_i}$ ▷ Update relative drift estimate
    end if
    $(\tau'_i, \tau'_j) \leftarrow (\tau_i, \tau_j)$
    $\hat{a}_i \leftarrow \rho_v \hat{a}_i + (1 - \rho_v) \eta_{ij} \hat{a}_j$ ▷ Update reference drift and offset
    $\hat{b}_i \leftarrow \rho_o \hat{b}_i + (1 - \rho_o) \left((\hat{a}_j \tau_j + \hat{b}_j) - (\hat{a}_i \tau_i + \hat{b}_i) \right)$
  end for
  Wake up again when $\tau_i = \frac{(k+1)T - \hat{b}_i}{\hat{a}_i}$
end for

6-4 Temperature Compensation

Already introduced in section 4-3, we will also include the pseudocode for the temperature compensation algorithm here for future reference. In Algorithm 5, T_t is the nominal turnover temperature, and h is the temperature dependency in $\text{ppm}\,^{\circ}\text{C}^{-2}$.

Algorithm 5 Temperature compensation, as executed by a node i

for every round k **do** ▷ After receiving messages
  Retrieve temperature T_i ▷ From measurement or memory
  $\xi_i(k) \leftarrow h(T_i - T_t)^2 \cdot T$
  $T_{i,k} \leftarrow T + \epsilon_i(k) + \xi_i(k)$ ▷ ϵ comes from the synchronization algorithm
end for

Chapter 7

Simulations

A closer look at the algorithm candidates' potential performance compared to Median was taken by simulation. A special-purpose simulator was built, and MemoryMedian, ATS and PISync were tuned and compared to each other on different networks and settings. This chapter will first treat the implementation of the simulator, and then present results from simulations in section 7-2.

7-1 Implementation

7-1-1 Introduction

The simulator is written in the Python 3.4 programming language, using the SciPy stack which provides routines for numeric and matrix-vector calculations (`NumPy`) and plotting (`matplotlib`). Python was chosen over previously used tools like Matlab and Octave to explore the potential of SciPy, and to see how well it compares to the established tools for these types of applications. Additional benefits are that it's free (both in the monetary and in the Open Source sense), and the blend of object-oriented programming and scripting lends itself well for simulation of communicating objects like nodes.

In the simulator, the progression of time is represented by the changing of each clock's phase. The process to replicate per node is one already familiar from chapter 5:

$$x_i(k+1) = x_i(k) + (a_i - 1)T + u_i(k)$$

Where $x_i(k)$ is node i's deviation from the reference time in round k, a_i is the drift factor of the node and T is the round time. In every round, a node can apply a clock correction $u_i(k)$, which is a function of the current and previous quantized time differences with its neighbours and information passed on in messages. For memoryless algorithms (Median), only information from the current round can be used.

In Figure 7-1, an outline of the simulator modules and their interaction is provided. The simulator is based on rounds. The active period of a round is simulated as if it were instantaneous; slots are not simulated, and the time differences are assumed to remain constant over the active period. An active period consists of the exchange of messages dictated by a certain topology. Afterwards, every node has a routine in which it executes the correction algorithm, and the clock is progressed to the next round. The join message and related

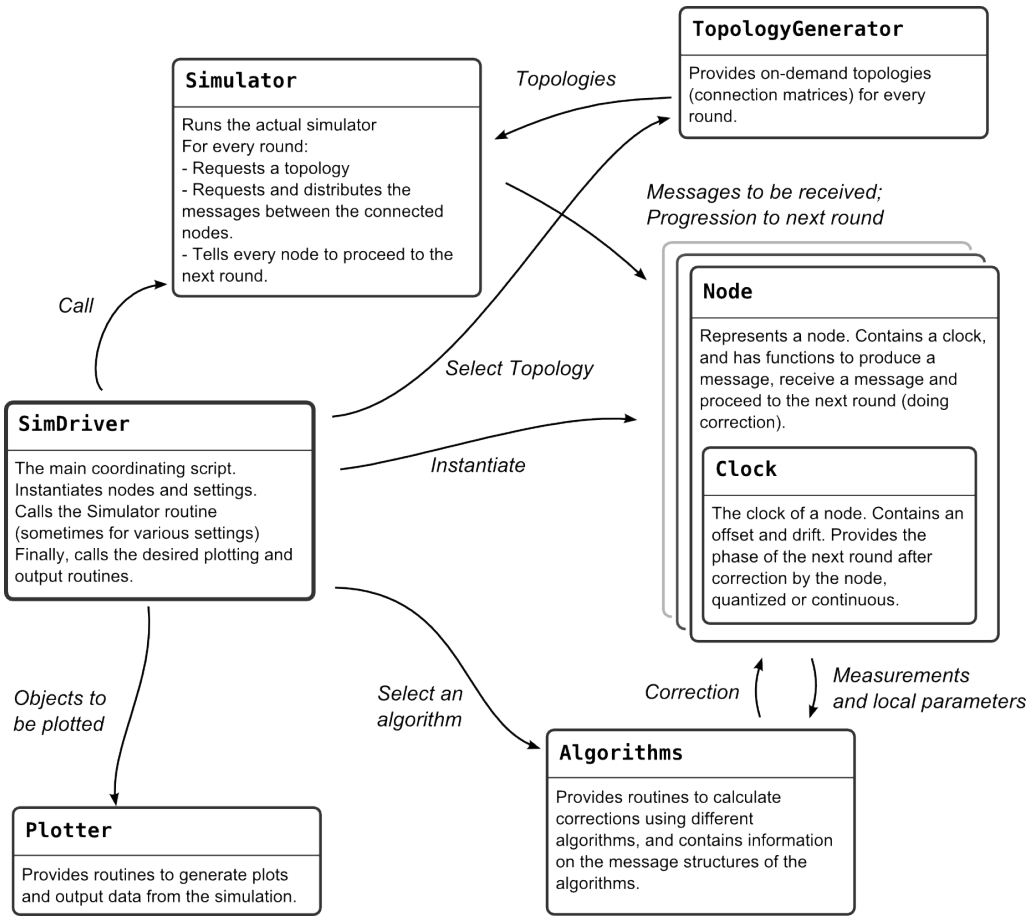


Figure 7-1: An overview of the modules that are used during a simulation.

functionality is beyond the scope of this simulator; the network can never dissynchronize, because communication takes place regardless of the time differences between nodes.

In the following, every module will be treated in detail separately, starting with the main simulator driver script.

7-1-2 Main Coordination

The execution of a simulation is done by running the `SimDriver` file. It is a Python script that sets up the settings for the classes (the optional quantization of the clocks, and the range of offsets for example), and instantiates N nodes (containing clocks).

When the nodes and settings are set up, the simulation is carried out for a prespecified number of rounds, by calling `Simulator.simulate` with the topology and the node list (the number of rounds is implicit in the number of topologies provided).

Finally, when the full number of rounds is simulated, the script calls routines from the `Plotter` to generate plots or output data. The data to plot is contained in local logs of the Node objects themselves, so just passing a reference to those object suffices. For example, to plot a graph of the time differences in the network, the list of nodes is passed, and the `Plotter` retrieves the data from the `Node` objects.

7-1-3 Simulation

The `Simulator` module contains only one method, `simulate`. By calling this routine with a topology generator and a list of nodes, the simulation is run (resulting in the logs of

nodes being filled). Every round a topology is requested from the `TopologyGenerator`. This topology, in the form of a directed adjacency matrix with ones and zeroes, is used for the routing of messages. For every node, the TX message is requested from the `Node` object with the specified ID (`get_TX`). This message is then fed to the receiving nodes (`put_RX`), which are inferred from the adjacency matrix.

After all the message exchanges are done, every node's `next_round` method is called, prompting them to calculate the clock correction from this round, and progressing the clock to the next round. When the topology generator runs out, the simulation is done.

7-1-4 Defining Topologies

The `TopologyGenerator` creates adjacency matrices for the network for every round. Although in reality the time differences influence the topology, it is assumed for now that the synchronization remains tight enough to provide no problems for gMAC. The possibility to generate topologies using time differences remains open in this architecture however.

There are roughly two types of topologies that can be generated: artificial and experimental ones. Experimental topologies are created from the logfiles of previous experiments. The number of nodes is fixed, but the number of rounds can be extended by bootstrapping (selecting randomly with replacement) from the previous sequence of topologies. Provided of course that the topology is static and the original number of rounds is large enough. Artificial topologies are created from theory, and can be as simple or advanced as desired.

Artificial

`random_topology(N, p, Nrounds)` The simplest of topologies is a directed Bernoulli random network: for `Nrounds` rounds, each of the `N` nodes has an independent propability `p` of connecting with a neighbour. By picking $\texttt{p} \approx 0.25$, the topology is somewhat realistic.

`gmac_fully_connected(N, Nrounds, Ns, maxSched)` In reality however, a tranmission either fails or succeeds, and if it succeeds, it reached all neighbours (as long as no schedules are active). This topology more like gMAC is produced by this method. It takes the number of nodes `N`, the number of slots `Ns` and a maximum number of schedules to add, and generates topologies for `Nrounds` according to gMAC logic. In Figure 7-2, these artificial topologies are compared to topologies that were measured in the experiments that we will treat later. For a fully connected, 10 node network the match is very good. For the 20 node network the simulated topology has less neighbours. The experimental network was spread out, so it was not purely a single hop network. The densities were lower, and thus the number of received messages can be expected to be little higher.

There is also a `_withpause` variant, that takes arguments specifying a moment and duration that no communication takes place.

`gmac_topology_from_static(A_static, Nrounds)` This method, used by the subsequent topology generators, creates a realization of the gmac protocol on the basis of a static A-matrix. This static matrix describes the topology of allowed connections. This method then figures out how many neighbours each node has, how many schedules it will thus run, and picks a random schedule to start on. Per round, every node picks a send slot, and based on the collisions, schedules and successful transmissions, an A matrix for the current round is created, and returned.

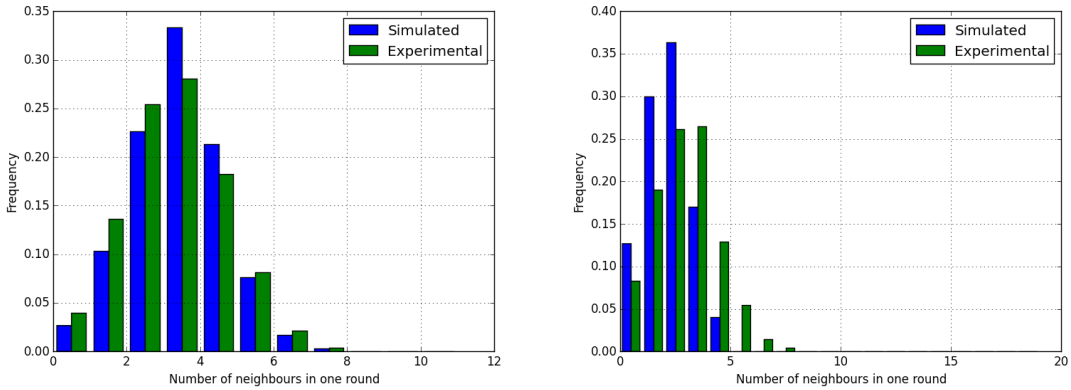


Figure 7-2: The histograms of number of neighbours per round over all nodes for experimental and simulated topologies of 300 rounds, with 10 nodes (a) and 20 nodes (b).

`grid_gmac(Nx, Ny, Nrounds)` Creates a pure grid topology (the 4-neighbour kind, not the 8-neighbour) of `Nx` by `Ny` nodes and feeds it into the gmac-realization method to create a sequence of topologies.

`random_geometric_gmac(Nnodes, Nrounds, average_neighbours)` The last of the artificial topologies is a random geometric network: a network where all nodes are scattered uniformly over an area, and have a fixed transmission range. Since the number of nodes within range is Poisson distributed (assuming a unit square as total area), a parameter for the average number of neighbours can be passed that defines the transmission range. Since the Poisson distribution holds for nodes on an infinite plane and this is only a unit square, the real average number of neighbours will likely be smaller. The random geometric graph is generated as a static topology, and converted into a dynamic gmac one per round.

Experimental

From previous experiments, log files are available which contain topology information. A logfile is a comma-separated file in which every line represents a transmitted message, with the following info: `frame number, sender, receiver, time difference`. By collecting all the sender and receivers for a round, and marking those combinations in a connection matrix, this information can be fed into the simulator. The experiments are of limited duration. If more rounds are requested, a bootstrap is done. For every extra round, a random entry from the basic dataset is drawn (with replacement) to serve as new data.

7-1-5 Nodes

The `Node` class represents a node. Every node is instantiated with an ID and a clock, which has a drift and offset as specified, or randomly taken from an interval if nothing is provided. Furthermore, every node has some local variables that may be used by the different algorithms. There are three main methods for a node:

- `get_tx`: this produces a message containing the node ID and the clock's phase for the current round, plus parameters required by the algorithm.
- `put_rx(msg)`: the message `msg` is received by this node. In this method the phase of the local clock is subtracted from the phase recorded in the message to get the

time difference (optionally lengthened by the transmission time estimate error). This value is saved for later use (plotting), and added to the message to be processed in `next_round`.

- `next_round`: at the end of each round's active period, this method is called to let the node proceed to the next round. The clock correction for this round is calculated (using the logic in the `Algorithms` module), and the clock's `next_round` method is called to compute next round's phase. Finally, the round number is increased and the message list is cleared.

7-1-6 Timekeeping

Every node has a clock object, that imitates an oscillator. All clocks share a frequency, round time (which must be a round number of ticks) and a setting for quantization or not. Clocks are instantiated with a drift and offset, which can either be specified or drawn at random from a range. Internally, the continuous time (`_phase`) is kept in microseconds to avoid floating-point inaccuracies. To the outside, times are communicated as seconds. To accomodate for the timestamping in ATS, a property that converts the round number and phase to a time is provided.

The only method of `Clock` is `next_round(correction)`, where the phase of the clock in the next round is computed using drift, offset, round time and correction. The phase for the round is stored for use in plotting later, and is made accessible to the `Node` for use in the TX message. If the quantize setting is on, the correction needs to be in whole ticks, and the phase exposed to other classes is rounded down to the nearest tick. The value of the offset determines the start time, so ticks do not happen simultaneously at clocks.

7-1-7 Synchronization Methods

The `Algorithms` module contains the logic of the different algorithms, their message structures, the values of the tuning parameters and one property which selects the algorithm currently used. Nodes call methods from this module to construct standard messages (`get_dict(algorithm)`) and to get the right correction (for example `get_median (message_list)`).

7-1-8 Quantization and Transmission Time Estimate

In subsection 5-1-4, the quantization and time inference are shown to lead to significant errors. To simulate this as well, a transmission time misestimation can be added by calling `Node.compute_transmission_time(message_size_bytes, slot_time_us, datarate_mbps)`, which performs the same calculations as are done by MyriaNeds node programming utility `makeNwParam` to get the estimated transmission time:

$$T_{transmit} = \texttt{int}(((T_{TXenable}[\text{ticks}] + T_{oa}[\text{ticks}]) + 1) \tag{7-1}$$

$$T_{TimeonAir}[\text{us}] = \frac{8 \cdot (1 + 5 + \texttt{message\_size}[\text{bytes}] + 2) + 9}{\texttt{datarate}[\text{mbps}]} \tag{7-2}$$

In (7-1), the time it takes to turn on the radio (specified by the manufacturer to within a microsecond) plus the time spent on sending the message, both expressed in fractions of ticks, are increased by 1 and cast to an integer, which is a round-towards-zero operation in C. Thus this combination is always sufficient ticks to transmit the message. The time to transmit, calculated in (7-2), is the total message size divided by the datarate. The message size contains the payload and some overhead (like the preamble and WSN domain).

If we want to include both quantization and transmission time estimate errors, we can do $\epsilon_q = q(\epsilon + (T_a - T_{oa}))$, since T_{oa} is a whole number of ticks by definition, and does not influence the quantization action. $(T_a - T_{oa})$ can be precomputed and added before quantization.

Both effects are not independent. If we still want to ignore the effect of transmission time estimate errors while keeping quantization, we could do $\epsilon_q = q(\epsilon)$. If we want no quantization but only transmission time errors $\epsilon_q = \epsilon + (T_a - T_{oa})$ would suffice.

Quantization has influence on more aspects. Every readout of a node's clock should be quantized, which is handled by the `Clock.phase` property. It checks the setting and returns the quantized value if needed. Internally, the `Clock._phase` variable keeps track of the continuous phase. Time differences recorded by the `Node.put_rx` method are quantized, and the correction applied in `Node.next_round` are whole ticks. To these ends, there is one method to quantize a clock value: `round_down_to_tick(phase, b)`.

7-2 Tuning

Here the results from the simulations done to get insight into different synchronization algorithms operating on MyriaNed are shown. Unless otherwise specified, the simulations are carried out with quantization and transmission time estimate errors enabled, a round time of 1 second, a 32 768 Hz oscillator, initial offsets between 1 and 20 ticks and drift values between −100 ppm to 20 ppm. First each algorithm will be tuned on a fairly simple network. Afterwards, the algorithms will be compared on various aspects. For a complete description of the implementation of each algorithm and the role of the different parameters, we refer the reader to chapter 6.

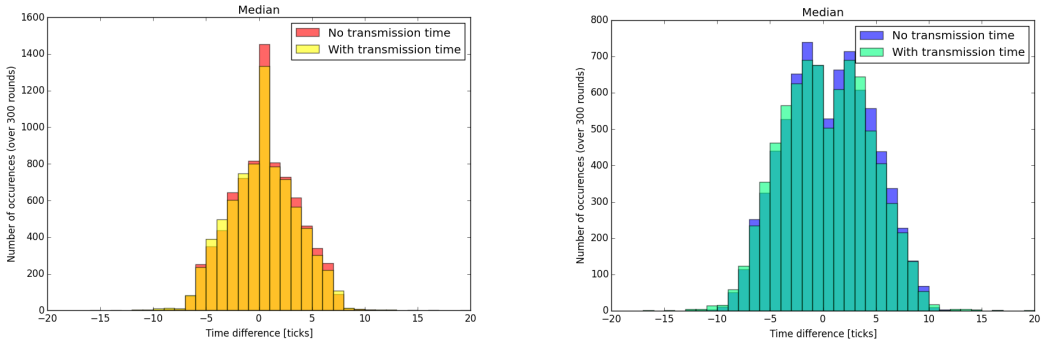
7-2-1 Median

To investigate the effect of different settings on the results of the simulator and to create reference results, we first present some simulations done with the Median algorithm. Throughout this section a topology of 10 nodes in a simulated gMAC-topology will be assumed, where all nodes are within each other's range. Although the topology is generated stochastically, the topology is the same for every experiment by resetting the random seed before every simulation.

Quantization and transmission time estimates

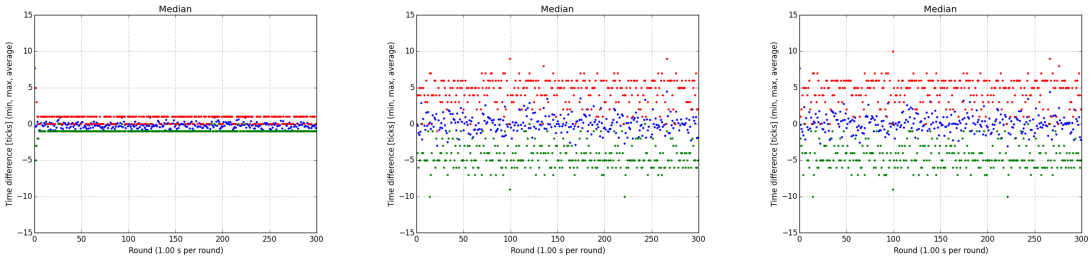
Apart from the clocks' own dynamics, the two most prominent sources of error are quantization and transmission time misestimations. In Figure 7-3, the four possible scenarios are displayed. There is virtually no effect of adding the transmission time misestimations when no quantization is present. The misestimation, which depends on the message size, is generally very small (about 3 µs for standard settings, up to tens of µs worst case) so this was to be expected. When quantization is present, this small effect leads to differences in the long run, because at a certain moment the small misestimation triggers a difference of a full tick. There are slightly more positive differences, and slightly less negative: the effect of a small, positive, constant misestimation.

The influence of quantization is more drastic. Because time readouts are always rounded down, the corrections are later and smaller than without quantization, allowing for slightly larger maximum errors overall, and a distinctive dip around 0 error. Because the gain is 0.5 and the correction is cast to an integer, errors of 1 tick do not lead to a correction.



(a) No quantization **(b)** With quantization

Figure 7-3: The median algorithm simulated on a 10 node network, round time 1 second, with and without quantization and transmission time misestimation. The histograms show the frequency of measured time differences.



(a) No drift, only offset **(b)** No offset, only drift **(c)** Drift and offset

Figure 7-4: Three simulations with the Median algorithm in a 10 node, fully connected network, with 1 second round time. Each experiment has the same topology, but different clocks. The maximum, minimum and average time difference recorded every round is displayed.

Only when a 2 tick error is measured, a correction is done. As such, if two nodes are in perfect sync, this will never be kept.

Drift and Offsets

In Figure 7-4, the effect of having clocks with offset, drift or both is shown. When only offset is present, the Median algorithm corrects quickly for these offsets (in about 5 rounds), and the time differences are kept within 1 and -1 tick in the steady state. The small remaining difference is due to the transmission time misestimation.

Drift is clearly harder to tackle. Since nodes accumulate a difference during rounds, the Median algorithm has to apply large corrections every round. When both effects are present, the initial offset is still corrected for, and the steady state is very similar to the drift-only situation. With the assumed drift values, we would expect drift-induced differences of up to 4 ticks per round. Because there is an initial offset to start with, and because not every round yields perfect correction, the actual maximum time differences can run up to 7 ticks.

Gain

Although the gain of 0.5 used in the Median algorithm is tried and proven, there might be a better value. A value higher than 1 is known to create an instability (at least in the case of two nodes), but anything in the range $(0, 1]$ might be suitable.

In Figure 7-5 some simulation results of varying this gain are plotted. The graphs again depict a histogram of the measured time differences during an experiment. The first one, depicting a generated gmac topology of 10 nodes within range, shows that low gains (0.1, 0.3) do not correct sufficiently strong, and lead to higher time differences. From 0.5 and higher, performance is good. The graph even suggests that a gain of 1 might perform even better. Intuitively, this might create an oscillatory effect of nodes simultaneously adapting to each other and doing the reverse in the next round. It seems that the randomness of communication in MyriaNed prevents this from happening – at least in this 10 node, well-connected network.

To test whether higher gains are indeed beneficial, the same simulation was done on a topology from an experimental log file, see Figure 7-5b. The performance is slightly worse for all gains: the peak around 0 is lower, and the tails are wider. The higher gains are still better than 0.5 however. A more challenging, realistic topology is depicted in Figure 7-5c, the two-group experiment. Again, all the results are worse (0.1 has been left out since it distorted the graph too much). The differences between the gains are smaller, but the higher gain proves to be the best again. This surprising result could be worth pursuing further.

7-2-2 MemoryMedian

The most straightforward alternative to Median is MemoryMedian, where the median time differences are integrated into a long-term correction estimate, and this is used for extra correction or correction when no neighbours are heard.

For MemoryMedian, there are three parameters to tune:

- $k_p \in (0, 1)$, the proportional gain. To make a fair comparison to Median, this was initially kept at 0.5. After the other parameters where tuned, changing k_p yielded no improvement.
- $\rho \in (0, 1)$ the parameter determining the lowpass behaviour of the error integration:

$$\alpha_i \leftarrow (1 - \rho)\alpha_i + \rho \ \mathrm{Med}(\theta_i(k)) \tag{7-3}$$

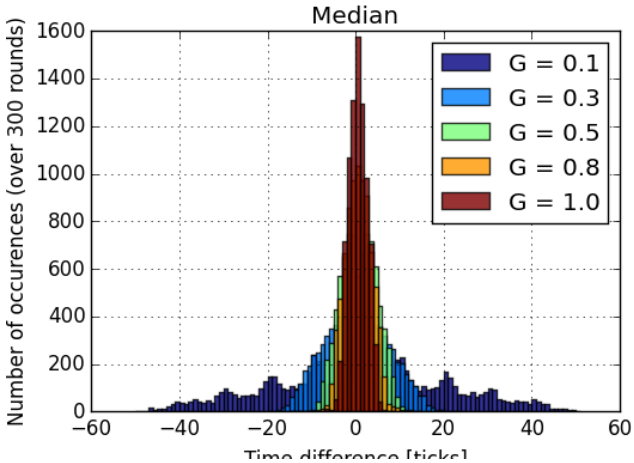
- k_i, the gain that is used to apply the error accumulation in the idle time correction:

$$T_{i,k} = T - k_i\alpha_i - k_p \ \mathrm{Med}(\theta_i(k))$$

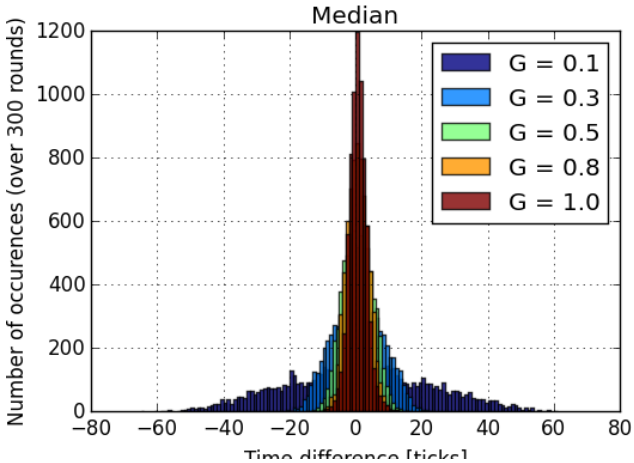
Initially a purely cumulative filter was used:

$$\alpha_i \leftarrow \alpha_i + \rho \ \mathrm{Med}(\theta_i(k)) \tag{7-4}$$

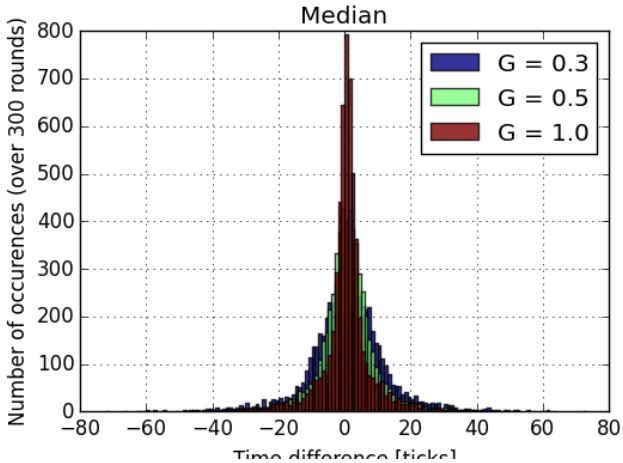
Theoretically, this filter would work better since it accumulates errors until the phase difference is zero, and then maintains that correct drift estimate. The alternative, Equation 7-3, needs a residual error to maintain its drift estimate, which will therefore never be perfect. The ρ value makes a tradeoff between adaptation speed and noise sensitivity of the integrated error. Since there is no noise in the simulator to deteriorate the estimate, the filter can be more aggressive in simulation than in reality. However, only median values are used for the estimate update, and the median already acts as a filter for noise and outliers. A relatively high value for ρ ($[0.5, 1)$) will most likely be acceptable in realistic circumstances as well. Since we still want to retain a memory effect, ρ should be sufficiently smaller than 1. A third consideration is, since the median is always a round number of ticks, ρ essentially controls the granularity with which the estimate of α is updated. The larger this granularity, the more inaccurate the estimate will be.



(a) An artificial 10 node gmac topology

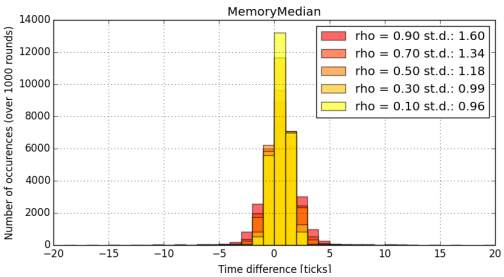


(b) A logged 10 node fully connected topology

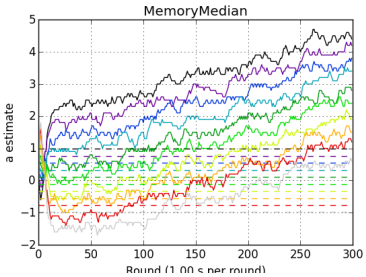


(c) A logged 13 node, two-group topology

Figure 7-5: The measured time differences over a 300 round simulation, round time 1 second, using the Median algorithm with different gains.

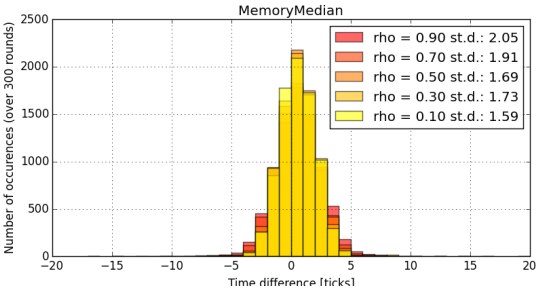


(a) A histogram of the time differences for different values of ρ ($k_i = 1$)

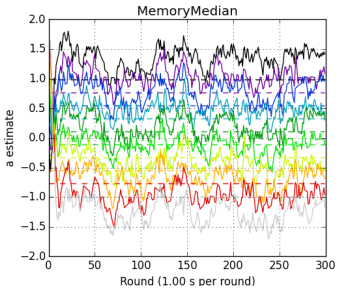


(b) The evolution of α_i with $\rho = 0.1$. Dashed lines indicate the values that correspond to the drifts used.

Figure 7-6: MemoryMedian in a 10 node gMAC topology, with 1 second round time and update equation (7-4)



(a) A histogram of the time differences for different values of ρ ($k_i = 1$)



(b) The evolution of α_i with $\rho = 0.1$. Dashed lines indicate the optimal values.

Figure 7-7: MemoryMedian in a 10 node gmac topology, with update equation (7-3)

Comparing Figures 7-6b and 7-7b, where the simulations were identical except for the update equation, this seems not to be the case. The first update equation does not settle on a perfect value but keeps increasing. The second update equation yields more stable estimates, but at the cost of a worse performance (Figure 7-7a). It seems from that Figure that ρ can be even smaller: the estimates overshoot and contain too many high frequencies.

In Figures 7-8a and 7-8b, even lower values of ρ are tried out. The performance stays roughly equal. When ρ is lower, the synchronization takes more time to compensate the comparatively large initial differences, and thus a slight increase in standard deviation is to be expected. The estimates displayed in Figure 7-8b give a better impression than before. We will stick with $\rho = 0.05$ for now.

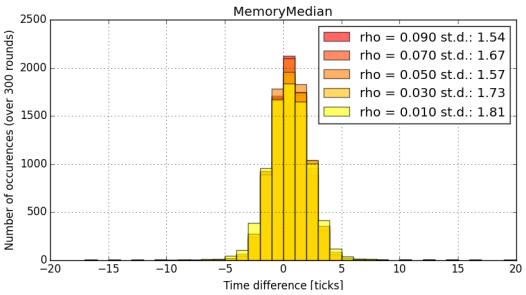
The value of k_i determines how strongly the error estimate is used for correction. If we trust the error estimate sufficiently, a value close to 1 could be used. Ideally, this would lead to perfect synchronization in the long term. Changes in drift values, drift noise, quantization noise and transmission time estimate errors will prevent this from happening.

The results in Figure 7-9 confirm this reasoning, as $k_i = 1$ indeed leads to the best results. Since $k_i = 0$ implies the unextended Median algorithm, this graph already indicates that MemoryMedian is indeed an improvement over Median.

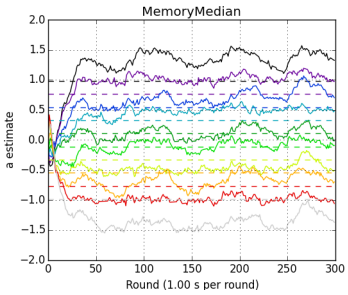
7-2-3 PISync

The next algorithm to compare is PISync, proposed by [75]. The algorithm uses a PI controller in every node for local correction, leading to global synchronization.

The parameters of interest in this algorithm are:



(a) A histogram of the time differences for lower values of ρ ($k_i = 1$)



(b) The evolution of α_i with $\rho = 0.05$. Dashed lines indicate the optimal values.

Figure 7-8: MemoryMedian in a 10 node gMAC topology, with update equation (7-3), and much smaller ρ values.

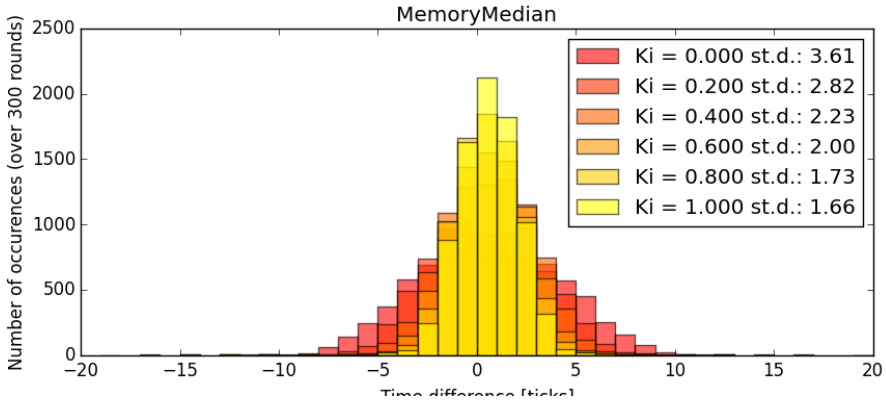


Figure 7-9: The performance of Memory Median in a 10 node gmac topology, 1 second round time, for different values of k_i.

- b: The gain with which to add time differences to the proportional correction:
- a_{min}, a_{max}, a_c: The gain used to integrate errors, either adaptive (using values between a_{min}, a_{max}) or constant (using a_c).
- e_{max}: the maximum error that is used for integration. Higher errors are considered to be caused by initial offsets or disturbances, and should not be used for drift compensation.

In [75], suggestions are done for tuning the parameters. By doing an assumption on the maximum drift difference in the network, e_{max} can be computed:

$$e_{max} \approx \frac{\Delta f_{max}}{f_n} T_f$$

Which is the maximum frequency difference that can occur ($\Delta f_{max} = \max(f_i - f_j)$), divided by the nominal frequency f_n times the frame time. For the assumptions in our model (maximum 120 ppm difference, 1 s frame time), this is ≈ 3.93 ticks, so e_{max} should be 4 ticks per 1 s frame time.

For b, a gain of 1 is said to provide best convergence results. However, this is when correction is done immediately after reception of a message. In MyriaNed, messages are accumulated. We could opt for b being 1 over the expected number of messages. This is dangerous, since if by chance the number of received messages is unusually high, the proportional correction will be as well. The most faithful reproduction would be to use the average phase difference multiplied by b.

The maximum value for a should be $a_{max} = \frac{1}{f_n T}$ for the fastest convergence of the local controller, as claimed in [75]. Taking $a_{min} = 0$, and $a_c = \frac{1}{2 f_n T}$, right in between seems sensible.

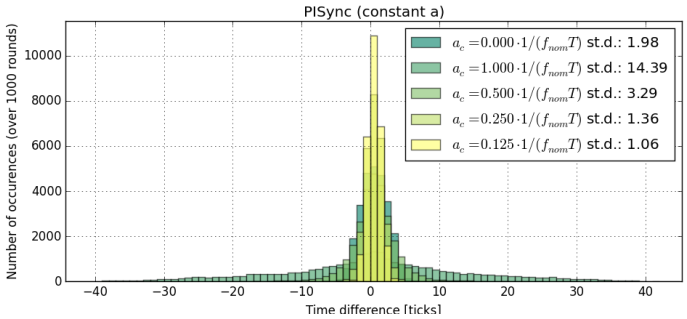
The graphs displayed in Figure 7-10 tell a different story. Starting out with the simplest scenario, Figure 7-10a shows the performance in the static case for different values of a_c. For $a_c = 0$ (only proportional correction), the results are reasonable. The errors increase unacceptably for $a_c = \frac{1}{f_n T}$, and only start to get better for much lower values. A value of $a_c = \frac{1}{8 f_n T}$ performs best (lower values were tried to no avail). In Figure 7-10b the adaptive version is tested. This indeed performs better than a consstant gain, and again needs a lower value than analytically expected. Perhaps the inaccuracies caused by quantization cause the estimates to vary too much with high gains, and a more conservative approach leads to better results.

The remaining two Figures 7-10c and 7-10d explore the settings for e_{max} and b. For e_{max}, 4 ticks seems about right, but one more or less is not critical. Even higher values were tried, with no dramatic effects. The value of b is more influential: a value of 0.8 or 0.5 is better than 1. Since the process is very much like the Median algorithm, it could be expected that the ideal value would be in this range. Perhaps it is again the quantization that make less aggressive values perform better than the theoretical optimum. We will stick with 0.8.

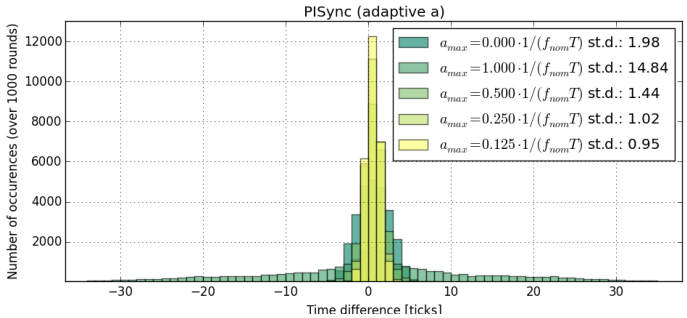
7-2-4 ATS

Average TimeSync (ATS) is another one of the fitting candidates from literature. It involves more analytic parameter estimation, and good experimental results are reported.

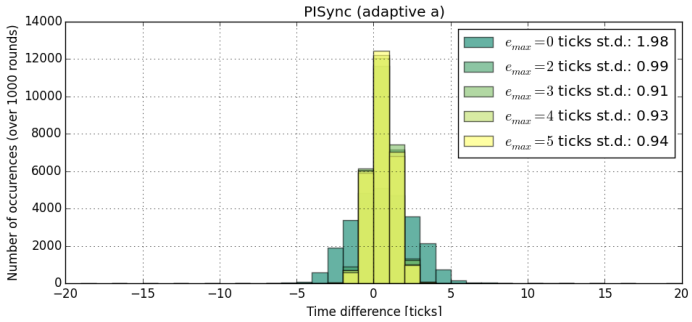
There are three parameters to be tuned in ATS:



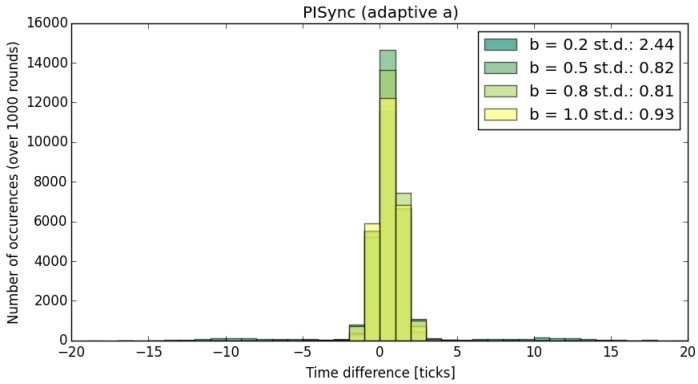
(a) a_c (nonadaptive), for $e_{max} = 4$



(b) a_{max} (adaptive), for $e_{max} = 4$ and $a_{min} = 0$



(c) e_{max} in the adaptive case, with $a_{max} = \frac{1}{8 f_n T}$.



(d) b in the adaptive case, with $a_{max} = \frac{1}{8 f_n T}$ and $e_{max} = 4$.

Figure 7-10: Tuning a, b and e_{max} for PISync, again on a 10 node, 1 second round time network.

- $\rho_\eta \in (0, 1)$: the low pass filter parameter for updating the relative drift estimate towards neighbours.
- $\rho_v \in (0, 1)$: the low pass filter parameter for updating the reference drift estimate.
- $\rho_o \in (0, 1)$: the accumulative low pass filter parameter for updating the reference offset estimate.

These three parameters are similar in function (the offset filter is accumulating instead of averaging is the only exception), and are of the same magnitude. The authors of the ATS algorithm intended both the offset and drift estimation to happen simultaneously. The estimates of η_{ij} have less input than the other two estimates, since they need multiple messages to be received. They could be tuned more aggressively to converge fast enough. On the other hand, outliers or errors could then have too big an influence. In the original paper Schenato and Fiorentin [57] propose the following gains: $\rho_o = \rho_v = 0.5$, $\rho_\eta = 0.2$. A later publication by other authors [28], where ATS is compared to another algorithm uses: $\rho_\eta = \rho_o = \rho_v = 0.5$. Note that – contrary to the formulation of MemoryMedian – a lower value for ρ corresponds to a faster filter. A quick test shows that $\rho_\eta = 0.5$ performs better than 0.2, so this will be used as a starting point.

In Figure 7-11, the three parameters are varied independently, keeping the others at 0.5. The influence of parameter values is very large: for some, the algorithm even seems unstable. For ρ_η and ρ_v high values (leading to conservative filters) are better. Since the drift to be estimated is a quantity very close to 1, a small step size makes sense. ρ_o can be more aggressive: lower values seem optimal. An iterative process, where every parameter in turn is varied in a range around its previous value and set to the best result leads to $\rho_\eta = 0.8, \rho_v = 0.9$ and $\rho_o = 0.05$. The drift estimates are slow, but the offset estimate corrects almost without memory. The results are good, with a standard deviation of the time differences of 1.17.

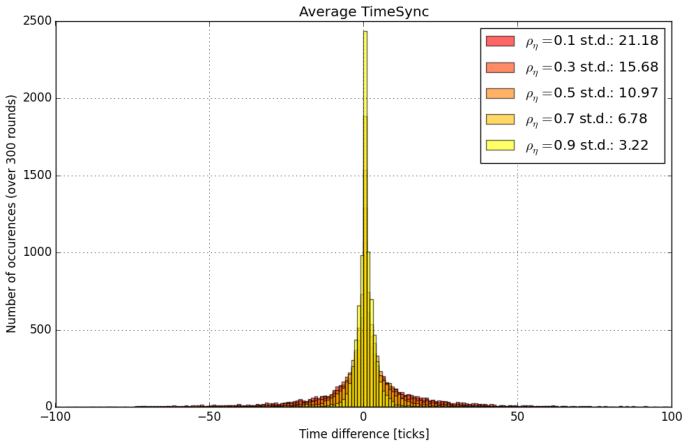
The parameter estimates produced by ATS are displayed in Figure 7-12. The consensus and estimation of $\bar{a}$ works very well, drastically reducing the drift variability in the network. The b estimates are much more variable (but we were not able to improve this with the filter parameter), yet this leads to the best synchronization behaviour. The cause of this variability is not entirely clear, but there is a tradeoff: to get good synchronization we want to act quickly on time differences (aggressive filter), but to reject inaccuracies and a more stable estimate we would want a slower filter. Apparently the first cause is dominant for performance. Removing the quantization and transmission time estimates leads to completely constant $\hat{a}_i a$ and $\hat{b}_i$ estimates for all nodes after about 100 rounds (graphs not shown).

7-3 Comparison

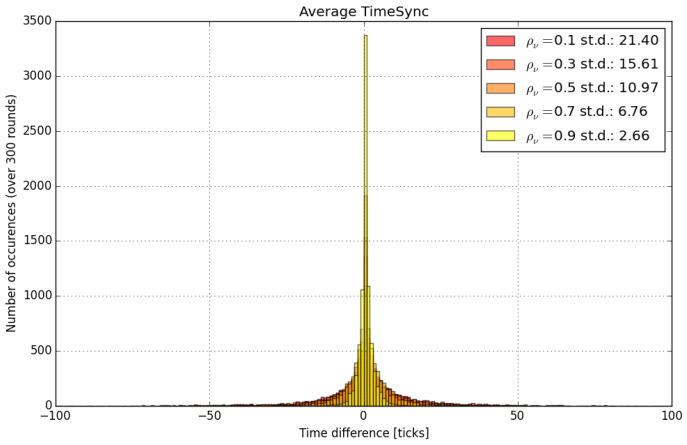
The four contending algorithms, with the parameters tuned to the optimal values, are compared in this section while varying different external parameters of influence: topologies, frame times, network size and density, their correspondence to external time and the recovery from extreme disturbances will all be treated in the following subsections.

7-3-1 Topologies

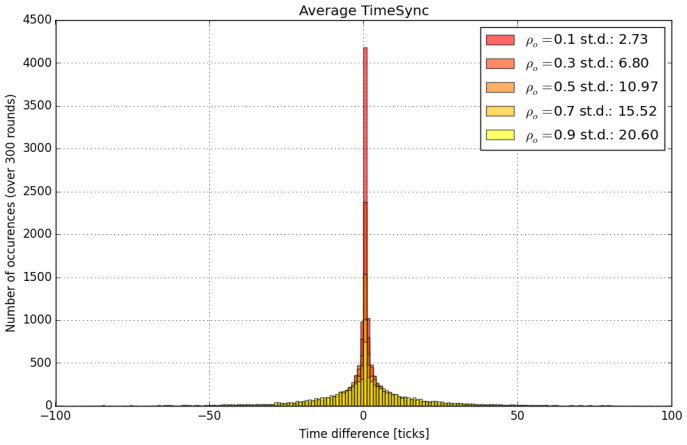
As mentioned in section 7-1, the simulator can generate several artificial topologies and read in previously retrieved experimental topologies. In this section those topologies will be fed into simulations using the different algorithms. The experimental data stems from small-scale experiments performed before this thesis work [35].



(a) Varying ρ_η

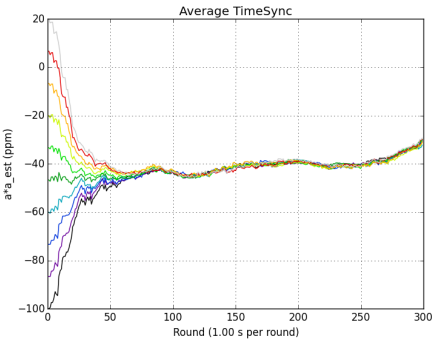


(b) Varying ρ_υ

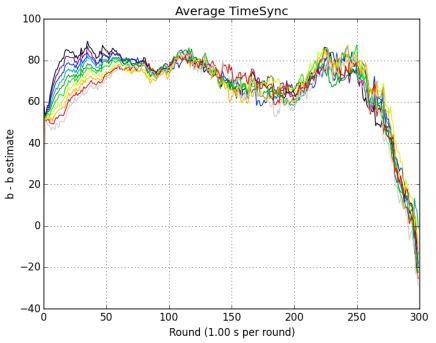


(c) Varying ρ_o

Figure 7-11: ATS on a 10 node gMAC network, 1 seconds rounds, with the tuning parameters varied.



(a) $\hat{a}_i \cdot a_i$ for all nodes, $\rho_\eta = 0.8, \rho_v = 0.9, \rho_o = 0.05$, which should approach the same value ($\bar{a}$) for all nodes.



(b) $b_i - \hat{b}_i$ (in ticks) for all nodes, $\rho_\eta = 0.8, \rho_v = 0.9, \rho_o = 0.05$, which should be driven to the same value ($\bar{b}$) for all nodes.

Figure 7-12: The parameter estimates of ATS.

10 Node Networks

First three different topologies with 10 nodes are compared, as displayed in Figure 7-13. All algorithms were tuned to perform optimally on the first topology displayed: an artificial gmac topology. The standard deviations are similar as in the tuning stages. Any differences with previous results stem from a different random seed leading to a different topology (but results in the same figure have *exactly* the same topology). It is clear that all three new proposals are an improvement over Median, but their respective performances vary.

The second figure displays a group of 10 from experimental data. All algorithms have more difficulty with this topology, perhaps because other factors (interference, radio propagation) influence the number of messages successfully received. Indeed, for the artificial topology the average number of messages received per node per round is 2.7, where it is 2.4 for the experimental topology, and 1.1 for the line topology.

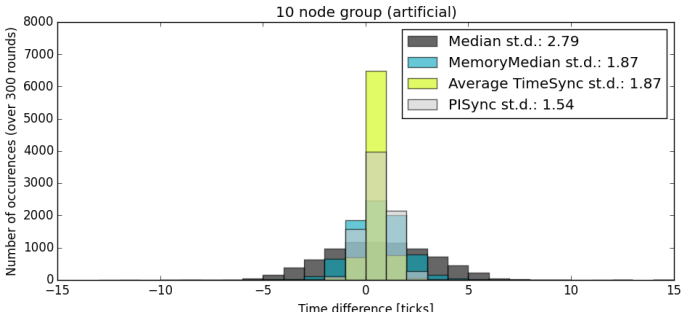
The third figure is a dramatically more different topology: a real world line topology. All distributions are considerably flattened out, and the maximum number of spot-on synchronizations (0 ticks difference) is less than a third of before. Strangely, PISync has a very bad performance suddenly, while MemoryMedian is the best in these circumstances.

Grid Network

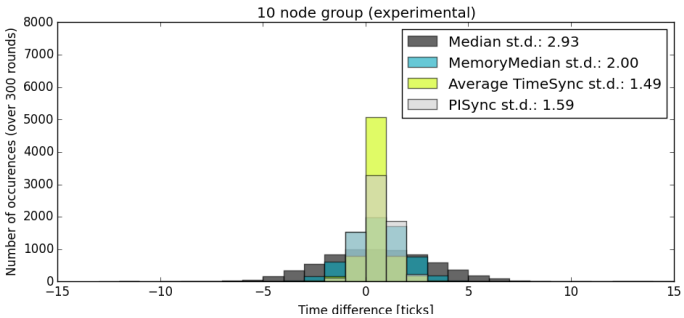
A canonical (but unrealistic) topology is the grid topology. Nodes are arranged in a grid, and can only communicate with their four orthogonal neighbours. This topology can provide insight in the per-hop deviation that an algorithm allows, and how large the time differences within a network can grow.

By simulating this topology with a gMAC process the results in Figure 7-14 were generated. In the meshes, every intersection corresponds to a node in the grid. The height of the intersection displays the average time difference with the node at (0,0) over the whole simulation. Drifts and offsets were randomly picked from the ranges mentioned at the start of this document, but were the same in each experiment (just as the topology). The histogram shows the time differences actually measured, so only between neighbouring nodes.

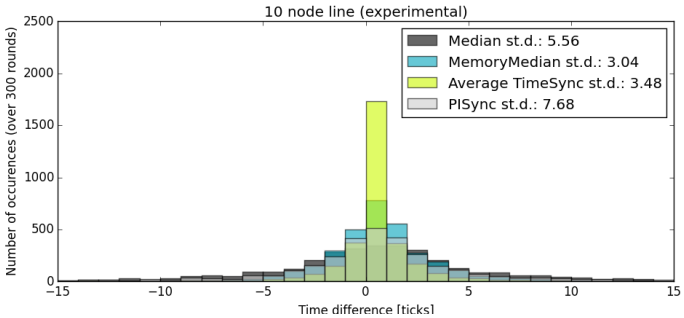
The meshes show the same pattern as in the first topology: all three contenders improve on Median. The drift estimation of MemoryMedian makes the differences of Median less pronounced, but an identical pattern is present. PISync and ATS are almost unbelievably



(a) In a group, artificial gMAC



(b) In a group, experimental data



(c) In a line, experimental data

Figure 7-13: Topologies of 10, 1 second round time, where the different algorithms are compared. Notice the much smaller y-axis in the third figure.

flat by comparison, with no average errors larger than two ticks, instead of the 15 or more ticks in the Medians. In the histogram, a different pattern is present: ATS is only slightly better (standard deviation-wise) than Median, and worse than MemoryMedian. Apparently the average (over rounds) difference between nodes is small, but its variance is large. The third graphs shows that this is because ATS has trouble converging at first. With so many hops in the network, the consensus processes take longer. Although the variance is large, the average difference with node (0,0) is not affected: ATS has similar internal time differences, but does a better job keeping all nodes at approximately the same pace.

Random Geometric Network

Another often-used topology is a random geometric network. Such a network is generated by randomly deploying nodes over an area, following a uniform distribution. The radio is then assumed to have a constant range, leading to a disc in which nodes are connected. This gives a network where the number of neighbours per node is Poisson distributed. In Figure 7-15, two realizations (with the same random seeds) are shown. In simulation, these 'allowed' connections were used to generate a gMAC topology, including scheduling in dense areas.

To arrive at a relatively topology-independent result, every algorithm was simulated on a variety of these random geometric networks. All the networks consist of 50 nodes deployed in a unit square, and by varying the radio transmission range, the average node degree was varied. On 120 different topologies (but again the same 120 for each algorithm), each simulation lasted 300 rounds. The standard deviations of the measured time differences are plotted in Figure 7-16.

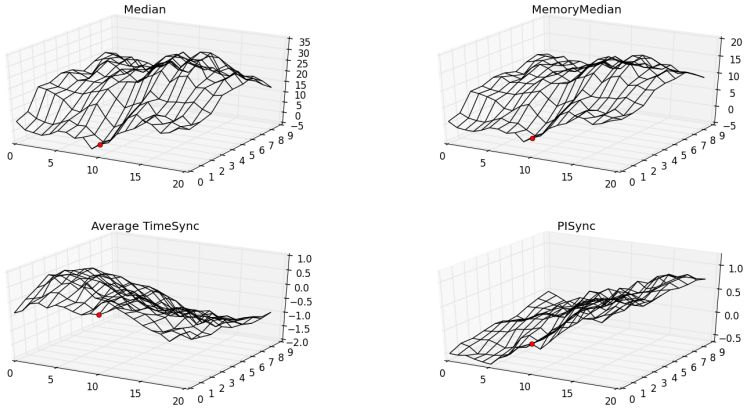
There is a large spread in values, showing that topology has a large impact on synchronization performance. To show the overall trend, a running average filter of 16 samples was applied, denoted by the solid lines.

The time differences are only measured between nodes that are connected. For very low densities, many nodes will be isolated (such as in Figure 7-15a), and will not contribute to the errors. This explains the generally low values for low densities. With increasing density (around 3 neighbours on average), more time differences are measured by all algorithms, but they seem to have trouble consolidating these differences. As density increases even further, the time differences decrease again. With more and more diverse neighbours, the algorithms get a less biased input, and use the extra information to converge more tightly. PISync performs best and is least affected by node density. ATS is good overall, but has some sudden bad performances. Median and MemoryMedian follow a very similar curve, with MemoryMedian being always slightly better.

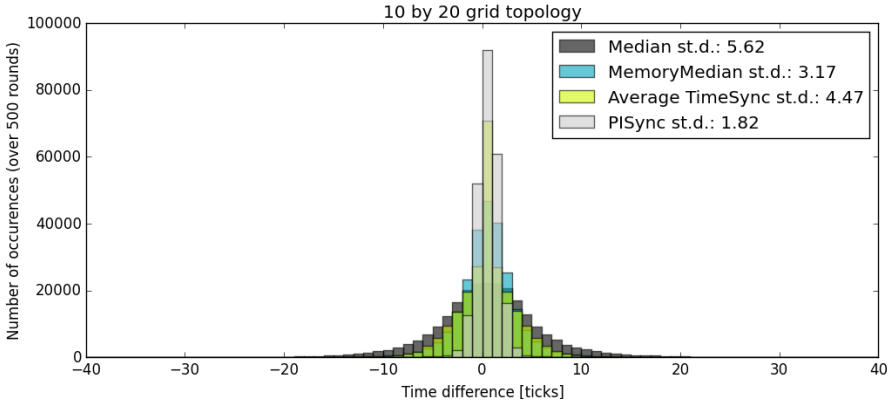
7-3-2 Frame Times

Synchronization algorithms should be able to perform with various frame times. For small frame times, measured differences and correction should be small, while long frame times lead to large drift accumulation and correspondingly large corrections. It is expected that drift estimation pays off especially for large frame times.

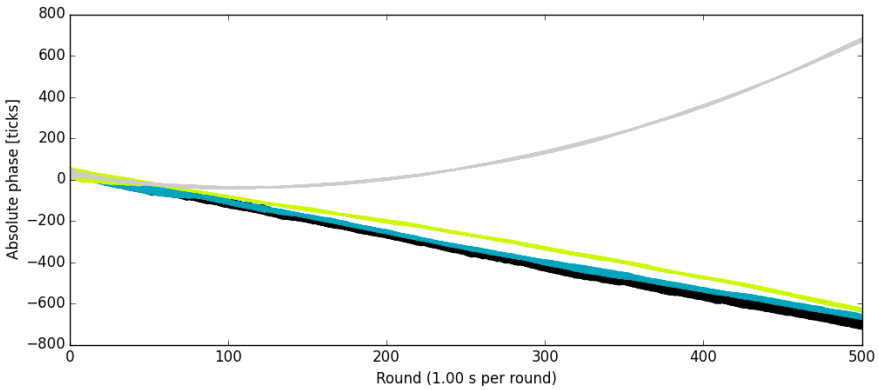
As before, simulations were carried out on numerous random geometric networks. For frame times between 0.5 and 10 seconds, 20 different random geometric networks with around 8 neighbours on average were simulated for 300 rounds for all four algorithms. The results, in terms of minimum, maximum and average standard deviations are displayed in Figure 7-17. All four algorithms seem to be roughly linearly dependent on the frame time.



(a) Average time difference with the node indicated by the red dot over 500 rounds, in ticks. Every coordinate corresponds to one node, drifts and offsets are distributed randomly. The z-axis has a very different magnitude for ATS and PISync.



(b) The corresponding histogram of measured time differences



(c) The phase of all nodes with respect to reference time. Colors are the same as in the histogram.

Figure 7-14: Results of the grid topology, 1 second round time.

(a) Average degree 4 **(b)** Average degree 10

Figure 7-15: The allowed connections in an arbitrary random geometric topology, for average degree 4 and average degree 10.

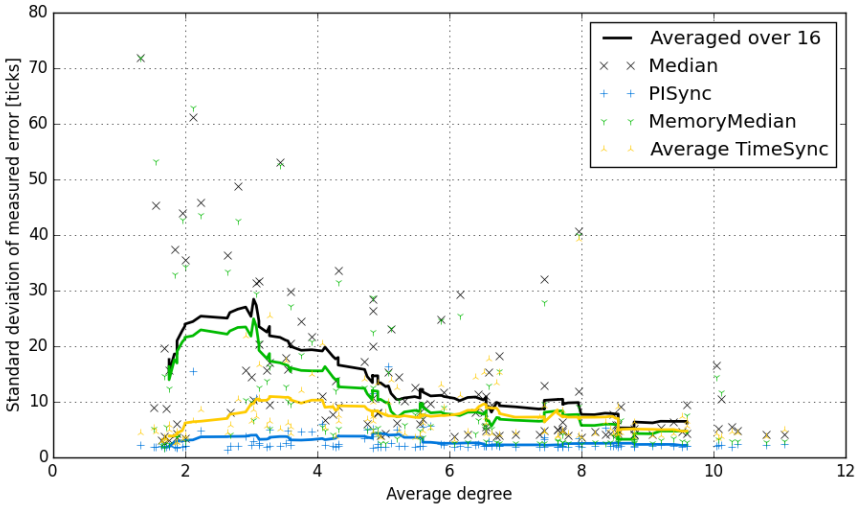


Figure 7-16: The time difference standard deviations for different topologies with varying average node degrees. Each marker represents a simulation, the lines are 16-sample moving averages over the data points to indicate trends. Every simulation lasted 300 rounds of 1 second, on a 50 node topology. The four algorithms were tested on the same set of topologies.

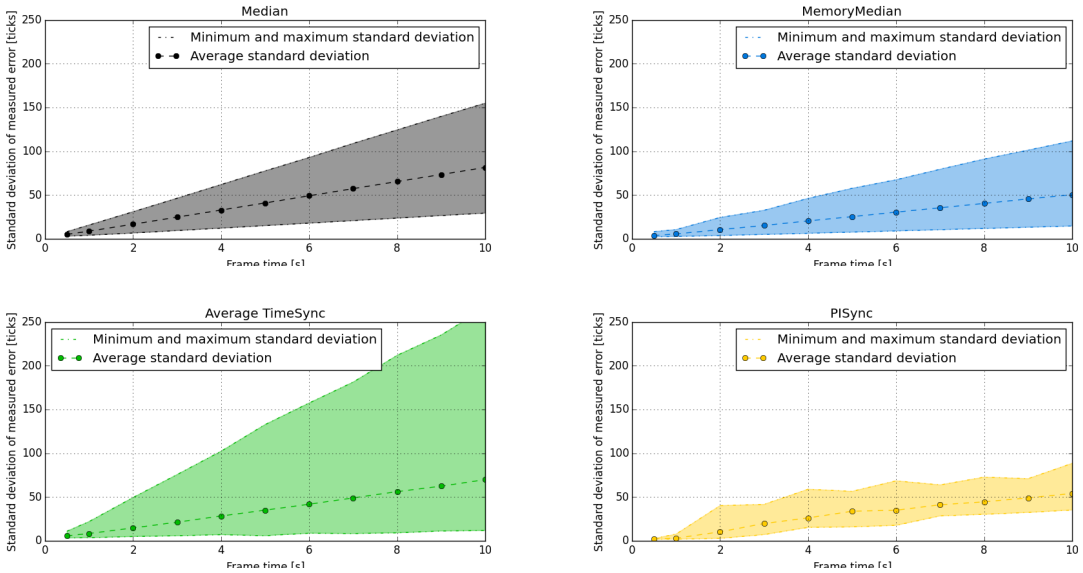


Figure 7-17: The effect of frame times on algorithm performance. Each algorithm was run 10 times for 300 rounds on a 50 node random geometric network with approximately 8 neighbours on average, for frame times between 0.5 and 10 seconds. The lines indicate the average standard deviation, and the areas the minimum and maximum standard deviation of measured time differences.

For Median this was expected: double the time to accumulate errors will lead to double the measured error. For drift-estimating algorithms it's somewhat surprising: apparently the drift estimation leads to a lower error variance, but still one that grows linearly with frame time. MemoryMedian and PISync are about equal in terms of average performance, but MemoryMedian has a larger diversity in standard deviations: it can perform better than PISync, but might just as well perform worse. The same holds for ATS, but in a very extreme way: its best runs are the best of all, but it has the worst runs as well; even worse than Median (although on average it is a little bit better).

The measured time differences – the synchronization error – is always an equilibrium between the accumulating differences making it larger on the one hand, and the convergence speed of the algorithms on the other. With increasing frame times, the accumulating error becomes larger, while the algorithms react in the same way. In Figure 7-18, the convergence of the algorithms for different frame times is shown by plotting the minimal and maximal time difference between neighbouring nodes over time. The simulations were started with a large initial offset in the range (-30, 30). For frame times of 0.5 and 1 second, the algorithms converge to a steady state within a smaller range than the initial offsets. The weakness of ATS shows: its convergence takes much longer than the other algorithms. The other algorithms converge faster and have the same steady state range. It shows that advanced algorithms do not have a clear advantage for frame times below 1 second.

For larger frame times the error is larger, as expected. Estimating drifts shows its advantage. MemoryMedian is consistently better than Median. Since there is not much convergence to be done, ATS does not underperform at the start, but it manages to converge in the long run, and reach the smallest error bound of the algorithms. PISync is very disappointing, and seems to increase its errors in the long run. It is possible that this is an effect of (not) tuning, since PISync's gains (e_{max} and α) are theoretically dependent on the frame time, but were kept constant here. These results seem strange compared to the findings in Figure 7-17, where PISync was the most consistent performer. Apparently,

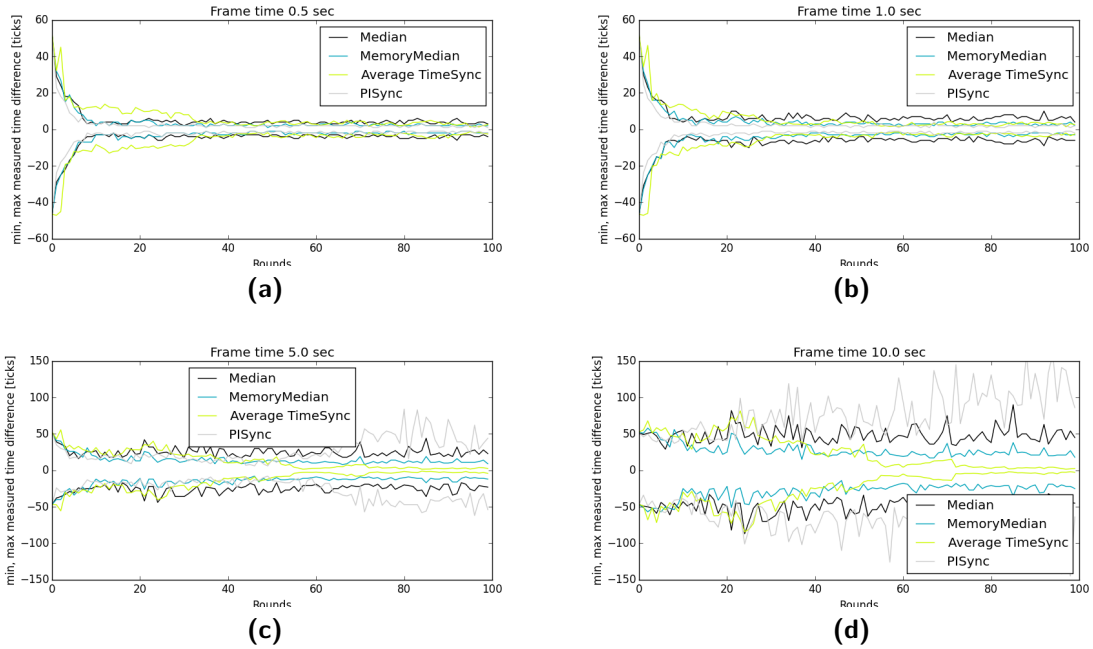


Figure 7-18: The minimum and maximum measured time differences in a random geometric network for different frame times, and different algorithms when starting with an initial offset evenly distributed over (-30, 30).

this specific realization is unlucky for PISync, and lucky for the others.

7-3-3 Network Size

Some authors (for example Fan and Lynch [18]) suggest that the network size or network diameter influences the local synchronization error negatively. To explore this for the different algorithms, simulations were carried out again on random geometric networks. For several network sizes from 10 to 500 nodes, 20 random geometric networks with average degree around 8 were generated. On these networks, a simulation of 300 rounds was run for every algorithm. The graphs in Figure 7-19 show the minimum, maximum and average standard deviation of measured time differences in the network.

The first thing to remark is that apparently the specifics of each topology are of larger influence than the network size. The average standard deviations seem to increase steadily with network size up to about 100 nodes, but the maximum standard deviations (worst cases) vary much more wildly. In short, the network size is not critical: if an algorithm can run successfully on challenging networks of 100 nodes, these results indicate it will not have more trouble with larger networks.

7-3-4 External Time

A secondary interest of the synchronization protocols is their correspondence with external time. Although a strict adherence to some external time standard is not needed (nor possible without communication), the deviation from 'real' time should remain reasonable. To gain insight in the algorithms' properties in this respect, simulations were run where the absolute phase was tracked. On a random geometric network, each of the algorithms was run for 300 rounds. In an attempt to control the deviation, simulations where one node does not correct its clock were run as well. These results can be used to assess the feasibility of equipping a single node with a very precise clock in order to make the whole

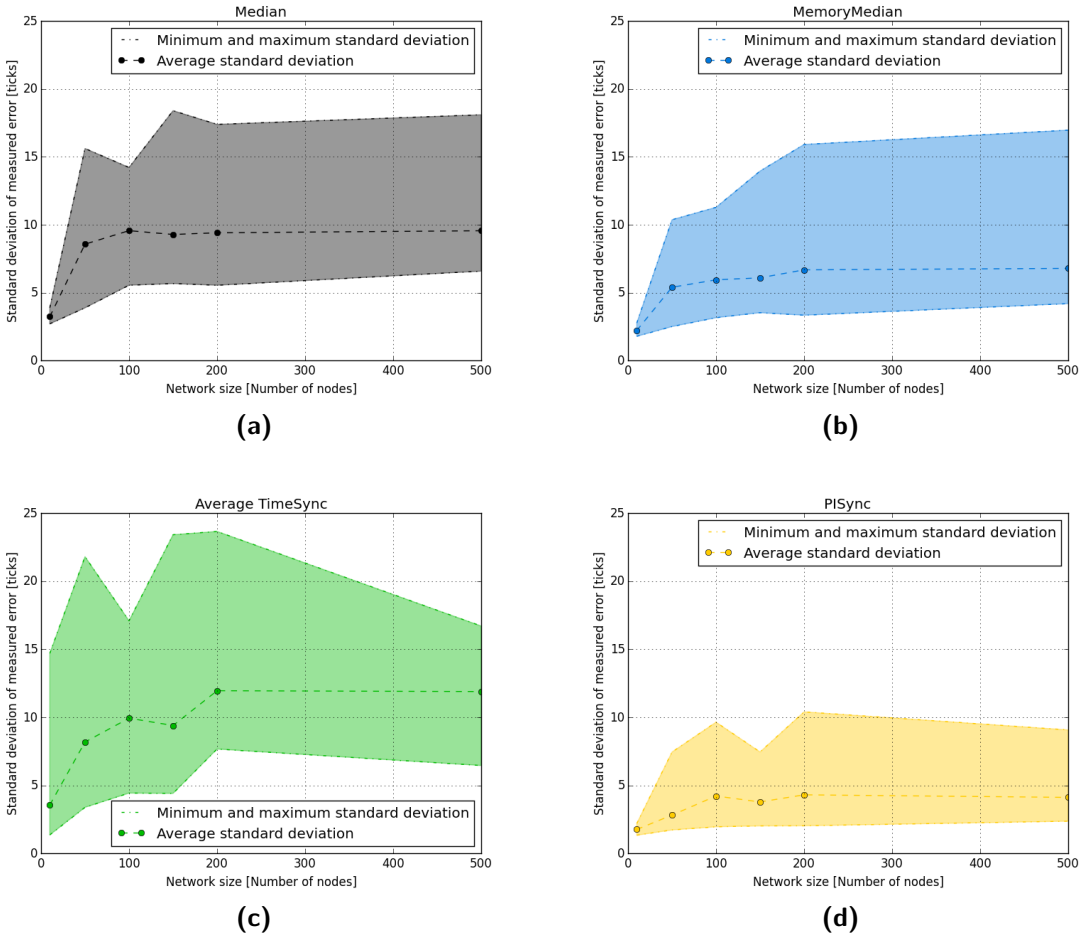


Figure 7-19: Standard deviations of time differences for various network sizes, 1 second round times.

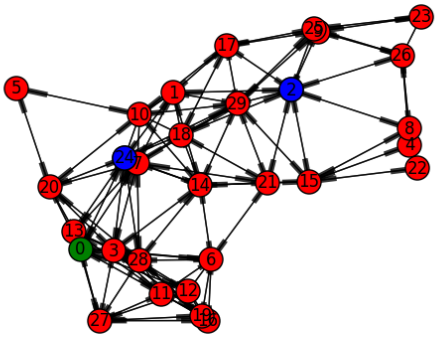


Figure 7-20: The network used for external time simulations. The green node (#0) is used as an anchor node. Nodes 2 and 24 (blue) are disturbed in a second experiment.

network synchronize with external time. In Figure 7-20 the (base) network used for these simulations is shown. In Figure 7-21 the results are shown.

Without anchoring Median, MemoryMedian and ATS have a drift that corresponds to the average drift in the network. PISync has a somewhat tighter line (less internal differences), but seems to continually speed up. If no countermeasures are taken, this can lead to a shortening of the network lifetime or even breakdown when the round time becomes too short for calculations.

With the use of an anchor node, matters only become worse. Each of the algorithms follows the anchor node, but reluctantly. The time differences within the network are considerably larger for all algorithms, and in practice this would prompt much larger guard times. With Median, it is clear that the anchor node has some periods where it is median neighbour to no one, and the network proceeds without taking the anchor into account. When it is a single neighbour by chance, it will trigger a large time shift, and drag most nodes with it. In MemoryMedian something similar happens. Because there is a lowpass-like behaviour in the correction that is applied, the disturbing effect is less on the short term but leads to a different pace on the long term. In a sense, the nodes' clocks have some inertia. ATS adapts its pace over time as well, but the time differences within the network are larger. PISync is similar to MemoryMedian: it follows the anchor but keeps a distance. Simulations have been done where the anchor node was given a positive drift, but this only worsened matters since the drift gap to bridge was even larger.

Preventing Speedup

In most cases, PISync performs best when it comes to the phase differences between nodes. In Figure 7-14 and Figure 7-21 it also becomes apparent that PISync continually speeds up: its drift estimate is increased indefinitely, making round times ever shorter. In practice this is unacceptable. Networks have to run for years on end and their frame time should be roughly equal to external time.

The cause for this can be found by considering the synchronization algorithm as a PI controller: there is a steady-state error that the controller can not remove, and the integral will wind up. In this case the error is caused by a small but systematic misestimation of

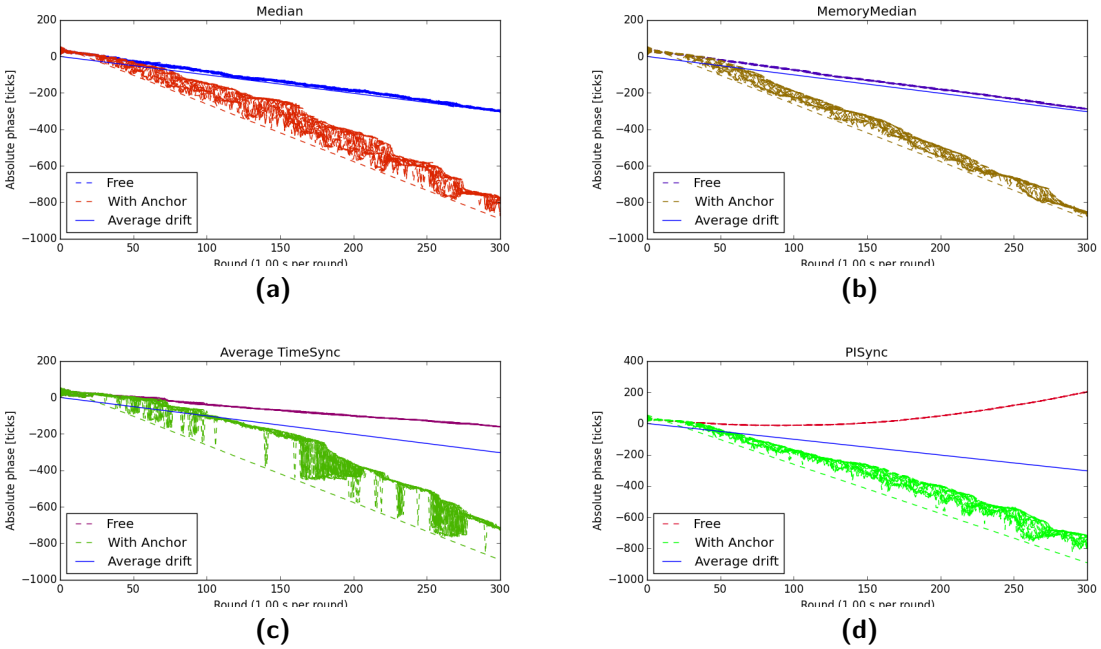


Figure 7-21: The absolute phase (with respect to ideal reference time) behaviour of the four algorithms. If the algorithm is run with an anchor, one node does not correct its clock in an attempt to dictate the pace of the network.

the transmission times. If we remove this misestimation from simulations, the speedup is gone as well, see Figure 7-22.

In the series of pilot experiments it quickly became clear that this 'speedup problem' was present in the real network as well, see Figure 7-23.

This Figure shows experimental results from a group of 10 nodes laid out on a desk, all around room temperature. This is not a very challenging situation, and we would expect at most 40 ppm drift between the nodes. The estimates by PISync keep on increasing far beyond that point, presumably because there is a small constant error in the transmission estimates. The measured time differences between the nodes remain small and consistent. In Figure 7-24, the receive time of messages modulo the frame time is plotted. Application messages are colored dots, and the time between active periods becomes a little smaller each round. After 1000 rounds, about 2.5 s are lost. The experiment was run for a total of about 5000 rounds, and the speedup was confirmed to continue all that time. This indicates the exact problem we were expecting: although the network itself is not hindered by it, the frame times become smaller and smaller.

In hindsight, this problem was not very unexpected. PISync integrates in the most naive of ways, by just summing a number derived from the measured time differences. If these measured time differences are non-zero even when the real time difference is zero (as is the case), then this integrated number will inevitably grow unbounded. A likely solution for the problem is thus to introduce a force that draws the integrated error towards zero, in the form of a filter.

We will now use the simulator to evaluate different options for this filtering. One possibility would be to use the same filter as in MemoryMedian:

$$\alpha_i(k+1) = (1-\rho)\alpha_i(k) + \rho\epsilon_i \tag{7-5}$$

A drawback of this approach is that it introduces a trade-off between having a strong memory, but not reacting very quickly (low ρ), or reacting quickly but forgetting easily

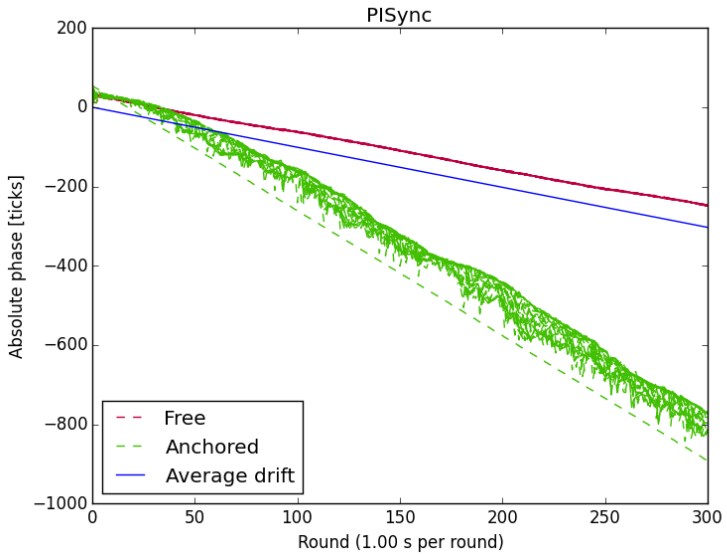
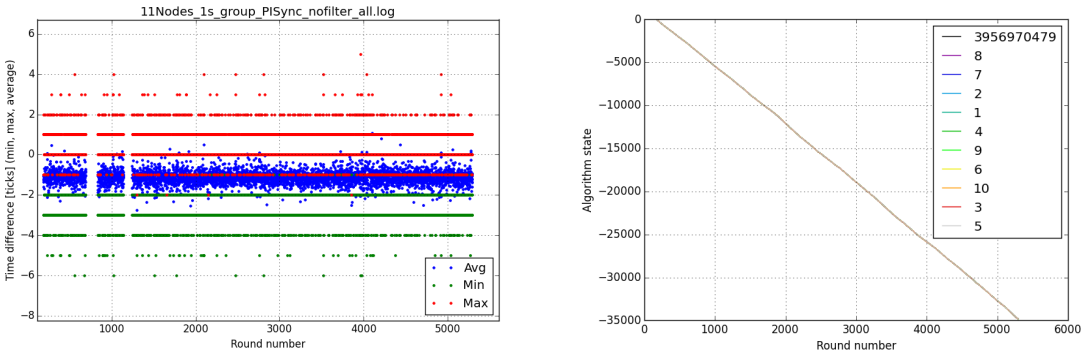


Figure 7-22: The experiment from Figure 7-21, only for PISync, without transmission time misestimation.



(a) Minimum, maximum and average time differences per round

(b) The algorithm state

Figure 7-23: Experimental results from a network of 10 nodes using PISync, at a frame time of 1 s. Clearly the filter values are decreasing all the time, while the time differences remain stable.

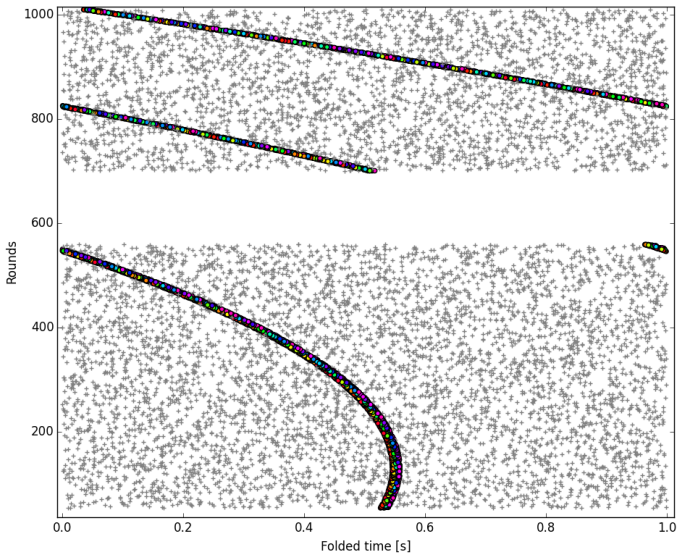


Figure 7-24: The timing of communications in a real life 10 node network using unfiltered PISync, at a frame time of 1 s. Coloured dots are application messages, gray crosses join messages. The empty space is a period in which the sniffer was accidentally turned off.

(high ρ). In the case of PIsync we want both, only not a memory that is so strong it grows unbounded. Another candidate solution could therefore be:

$$\alpha_i(k+1) = \kappa\alpha_i(k) + \epsilon_i \tag{7-6}$$

Where the integral value is reduced by a factor κ every update, but the new value is added as a whole. For the sake of easy referencing we'll name this the cumulative filter. Note that we mentioned a cumulative filter relating to MemoryMedian as well, but there the weight of $\alpha_i(k)$ was 1.

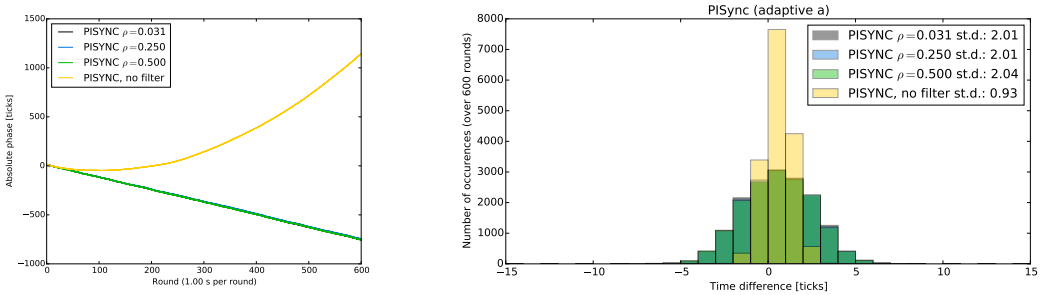
To assess the effect these filters have on PISync performance, we did some simulations. In Figures 7-25 and 7-26 the results are shown. The filter from (7-5) solves the speedup problem, regardless of the filter parameter. In all cases however, the performance was considerably worse than without the filter. This degrades PISync from a serious contender to an average performer, so we will not consider this filter for experiments.

The second filter is more useful. With different values for κ, the effects of the speedup problem are lessened, at the cost of a reasonable performance loss. A good tradeoff seems to be $\kappa \approx 0.97$.

7-3-5 Recovery from disturbances

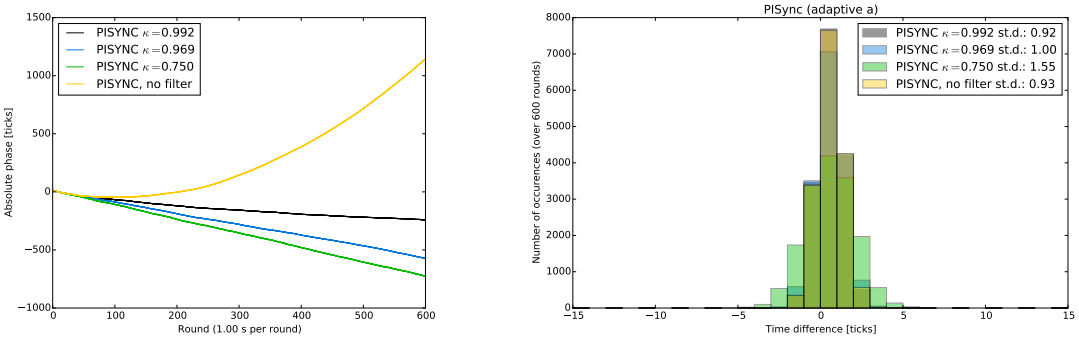
A final comparison of the algorithms was done with disturbances. To create something similar to a step-response, the same network as before (external time comparison) was run, but at round number 50, node 0 suddenly shifts 100 ticks backwards in time. In round 200 node 2 and 24, both very centrally located (Figure 7-20) are shifted +200 and -200 ticks respectively. The results are displayed in Figures 7-27, 7-28.

A disturbance by a single node has only small effects. In Figure 7-27 the responses of the different algorithms in terms of the phase with respect to reference time are plotted. The disturbance is added to the phase after a round, and the next round's phase is recorded after correction, so the visible portion of the disturbance does not equal 100 ticks. The



(a) Absolute phases **(b)** Histograms of time differences

Figure 7-25: Simulation results of PIsync with the filter from (7-5). Introducing the filter solves the speedup problem, but at the cost of a large performance penalty in all cases.



(a) Absolute phases **(b)** Histograms of time differences

Figure 7-26: Simulation results of PIsync with the filter from (7-6). There seems to be a trade-off between performance and speedup prevention here.

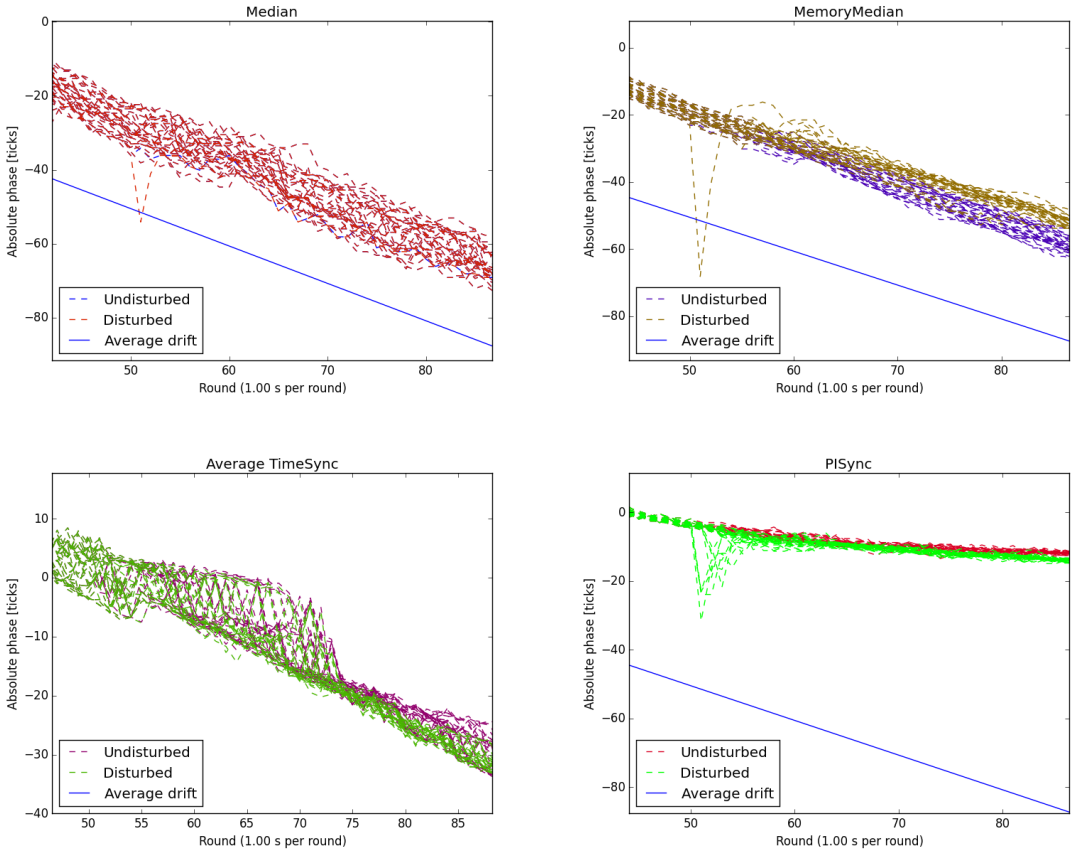


Figure 7-27: The response of the network to one node shifting -100 ticks at round 50, with respect to external time.

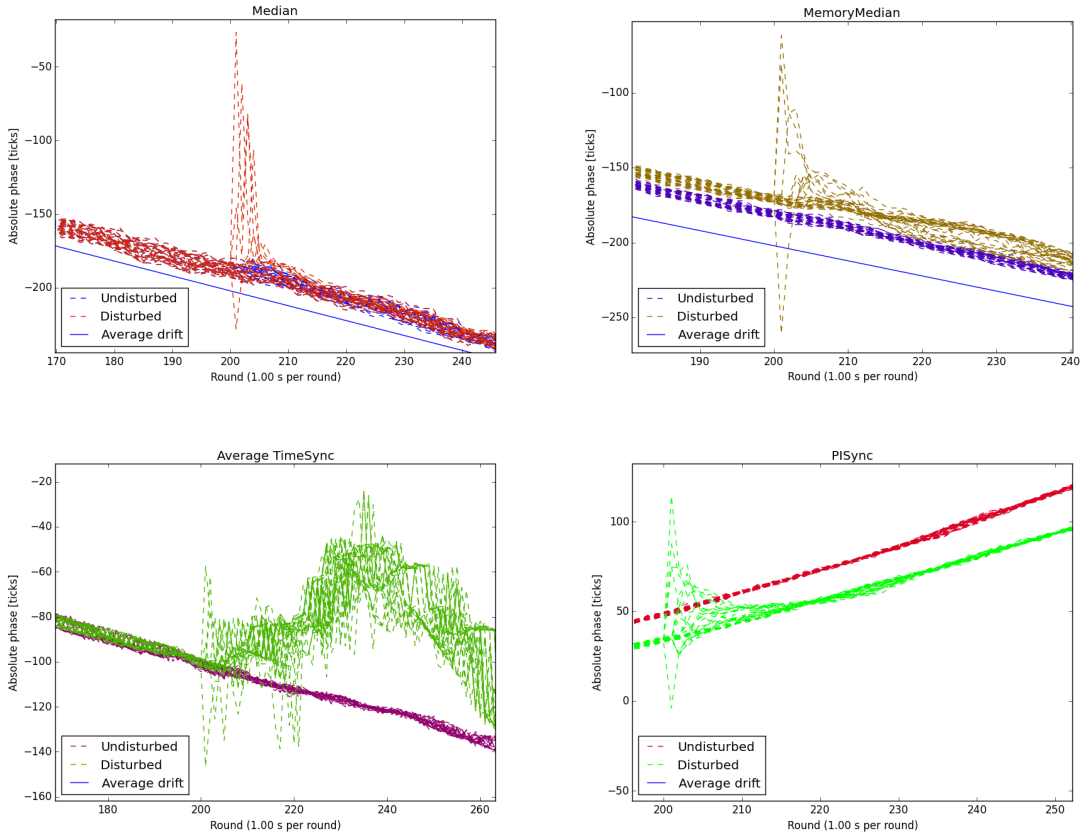


Figure 7-28: The response of the network to two nodes shifting -200 and +200 ticks at round 200, with respect to external time.

Median algorithm rejects the small disturbance completely: there is one round in which the phase of the disturbed node is large, but it is not a median for any neighbour, so the rest of the network proceeds exactly as without the disturbance. MemoryMedian reacts more. Due to the inertia in the protocol, the disturbed node takes longer to return to normal values, and has some overshoot because the node integrates its previous corrections. When it is close to the other nodes, it is the median neighbour of some, and the disturbance has a lasting influence in the network: it speeds up a little. Internally, there is little effect: the undisturbed nodes remain within a tight bound. The effect on PISync is more pronounced. Because the proportional correction of PISync is more aggressive and uses information from all neighbours, the disturbed node instantly drags some neighbours with it. ATS does not show any larger errors due to the disturbance, and even gets a little better after round 75.

The larger disturbances are more severe. Median again reacts best, but the disturbed nodes have some influence on others, leading to a brief period of large time differences. Not every node in the network is effected heavily, and the effect on the network as a whole is small. The reaction of MemoryMedian is similar to the previous one, but of larger magnitude: the disturbed nodes take longer to adapt back, overshoot a little and drag several other nodes with them. PISync does similar as before, but a slow oscillation is triggered that dampens out only slowly. ATS can not handle disturbances this large. The network is swept out of equilibrium and creates large time differences in the network, that do not settle anymore before the end of the simulation.

In Table 7-1 the (manually read) settling times are recorded. The drift-estimating algorithms take longer to settle since they are not memoryless.

Table 7-1: The approximate settling times of algorithms after disturbance. After the noted number of rounds the internal time differences are back to the level of before the disturbance.

Algorithm	Settling time [rounds]	
	Single small disturbance	Dual large disturbance
Median	3	10
MemoryMedian	9	15
PISync	13	30
ATS	1	...

A last disturbance is one that is less severe, but more realistic: no communication at all for several rounds. The results are show in Figure 7-29. Median – not having any memory – represents the largest time divergence. MemoryMedian keeps the errors smaller, but follows a very similar trend. It's estimation capabilities are not so strong. ATS shows what it does well: the drift estimation is spot on, and the situation does hardly worsen during the second silent period. The first pause comes too early, and the error buildup takes a while to be resolved again. PIsync is somewhere halfway when it comes to error buildup, but is the fastest one to converge to within reasonable bounds after the large period of silence.

7-4 Conclusions

Based on the simulations carried out, we can draw the following conclusions:

- Improvements to the synchronization performance must come from handling the drift and/or quantization, since these two effects currently have the largest impact. A transmission time misestimation of the size currently encountered has no dramatic consequences, and initial offsets are easily solved by Median already.
- The naive way to predict time differences based on a drift range are too optimistic: the measured time differences tend to be larger when no drift compensation is employed.
- Median seems to perform better for slightly higher gains than 0.5. This was on a relatively easy network however.
- In MemoryMedian, the best drift estimation does not lead to the best performance. A better (more stable) estimate is preferred however, since a noisy and easily-changed estimate could lead to strange results in difficult topologies.
- ATS has the highest potential, but also the highest sensitivity: slowest convergence, varying performance, sensibility to disturbances and parameter values. It has both the best and worst results, but is fragile. Moreover, it is the only algorithm that requires extra message content to be sent.
- MemoryMedian and PISync both improve on Median. PISync achieves lower time differences – both between neighbours and over the whole network – but is less robust to disturbances, needs different settings for different frame times and has a tendency to speed the network up. MemoryMedian offers the least improvement but seems to be most robust as well.

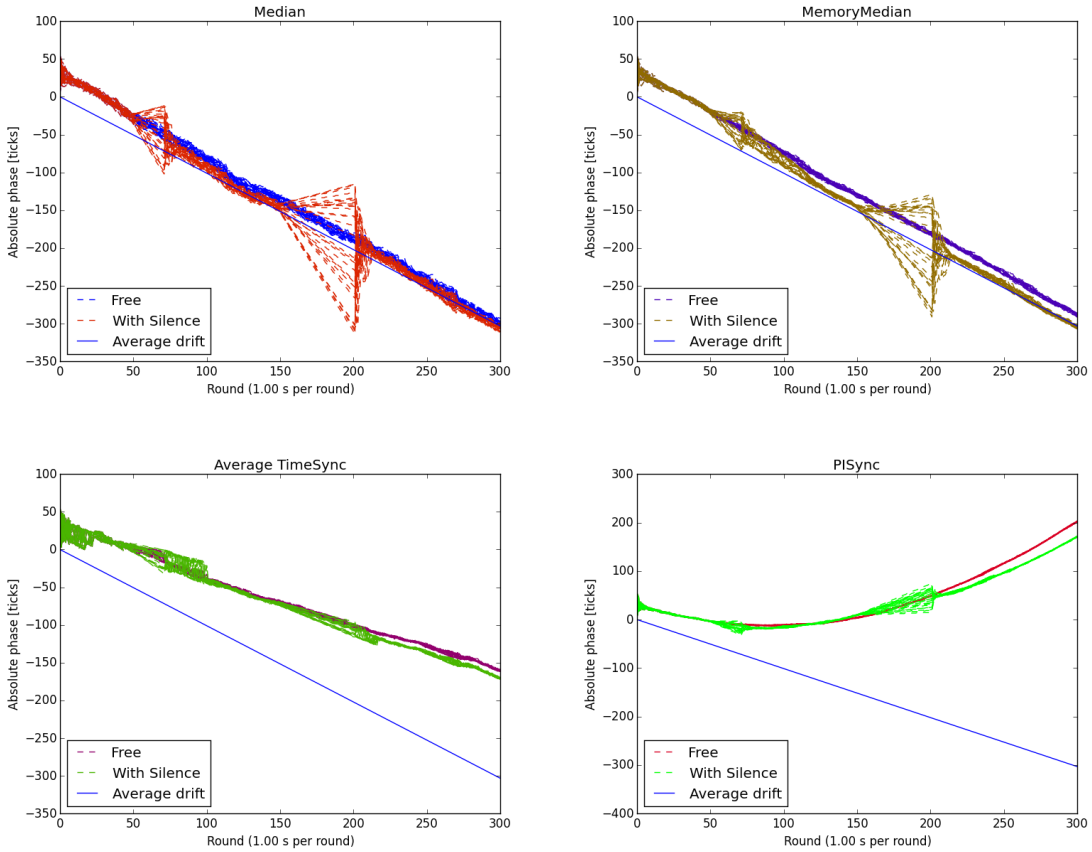


Figure 7-29: The experiment from section 7-3-4, but in round 50-70 and round 150-200 there's no communication.

Table 7-2: The algorithms scored against one another on the different comparisons done in this chapter.

Algorithm	Topologies	Frame Times	Network Sizes	External Time	Disturbance
Median	- -	-	○	○	++
MemoryMedian	○	○	+	○	++
ATS	+	- -	-	○	- -
PISync	++	+	++	-	○

Chapter 8

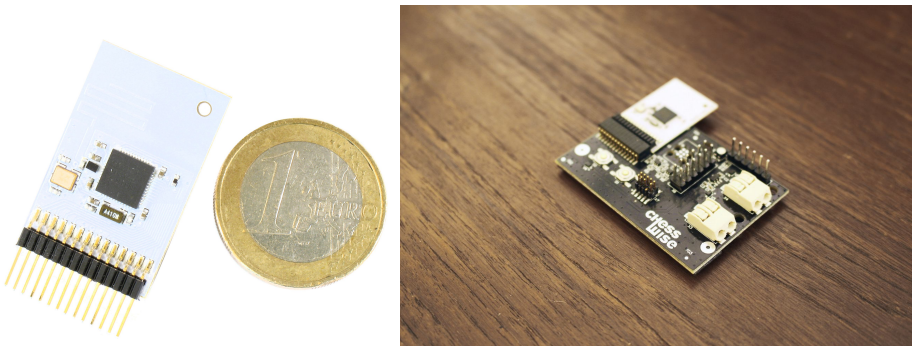
Experiments

To test the validity of the insights and conclusions drawn in previous chapters, experiments are invaluable. To complete the work of this thesis, several experiments on real life Wireless Sensor Network (WSN)s were done. This chapter describes the setup and implementation of these experiments, and shows the results. In section 8-4, the experimental results are compared with the results from simulations.

8-1 Setup

8-1-1 Hardware

Experiments were carried out with the wireless modules used for in-house development at Chess Wise. These modules consist of a Nordic nRF51822 chip and the bare minimum of extra hardware: some capacitors, resistors, two crystals and an antenna, see Figure 8-1. Via a set of pins these modules can be places on carrier boards, which provide power (via batteries or wires), a LED and connectors for programming and debugging.



(a) **(b)**

Figure 8-1: A MyriaModule, compared to a Euro coin (a), and on a standard carrier board, providing a power interface, LED and connectors for programming and debugging (b).

The Nordic nRF51822 has an ARM Cortex-m0 processor at its core, and is programmed in C. The 32kHz oscillator is a Seiko Epson FC-255, which has the same specifications as assumed in section 4-3 [58].

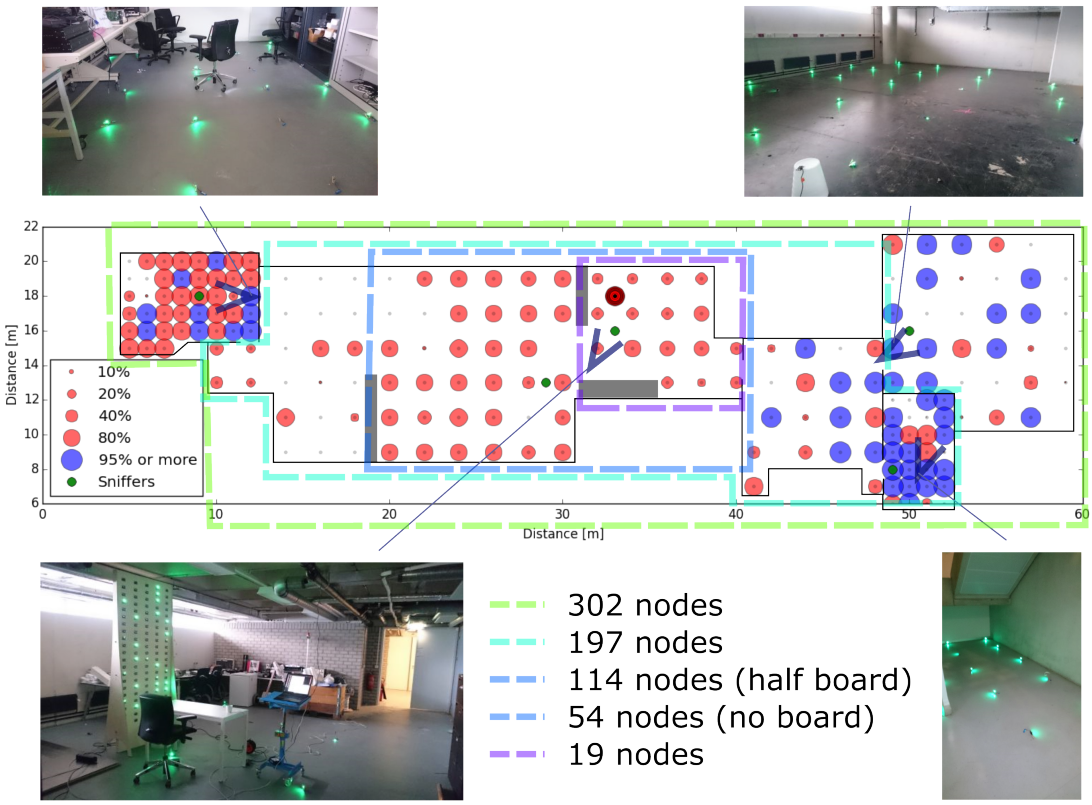


Figure 8-3: A map of the large scale experiments, with pictures of some regions. Sniffer locations and reception ratios are shown. Due to limited sniffer coverage, the number of nodes that was sniffed is slightly lower than the number that was present. Some of the nodes are visible in the pictures by their green light.

Figure 8-2: The climate chamber used for the temperature experiments. Some nodes are visible inside the chamber and on the ground in front of it.

For the experiments involving a controlled ambient temperature, a climate chamber was used, depicted in Figure 8-2. The machine can bring the temperature to levels between $-30\,^{\circ}\mathrm{C}$ and $120\,^{\circ}\mathrm{C}$. The operating conditions for the nRF51822 are between $-25\,^{\circ}\mathrm{C}$ and $75\,^{\circ}\mathrm{C}$ [45], so the whole range is covered by this instrument. Wireless transmission through the metal walls of the climate chamber is somewhat hampered, but enough messages come through to form a network. To have sufficient coverage with the data gathering (see subsection 8-1-2), a sniffer was placed both inside the chamber (via a cable hole) and outside.

The large-scale experiments were done in a large basement beneath Chess Wise' offices, see Figure 8-3. Nodes were spread out in a grid over the ground with distances of 1 or 2 meter between them. In the center of the basement we set up a large wooden board with a maximum of 105 nodes, normally used for demonstration purposes. Hopefully, the large network will have dense as well as sparse neighbourhoods.

8-1-2 Data Retrieval

For retrieving this data, two possibilities were considered: local logging or over-the-air reporting. Local logging guarantees complete data collection, but has disadvantages as well: the hardware to save data to

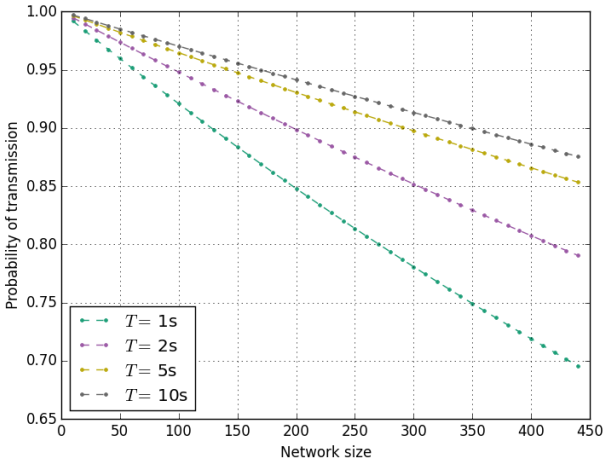


Figure 8-4: The expected success rate of join messages for a single sniffer over a one-hop network of different sizes.

has to be bought and interfaced with, and reading out all the data after the experiment means extra work and a slower iteration speed.

The alternative is to send the data contained in the 'join' messages of the network. This method is often used within Chess Wise to retrieve data from the network without interfering in its normal operation. The join mechanism is not influenced by it either. As long as the network under test remains reasonably synchronized, none of the nodes will be listening during the join messages. Any information contained in these messages will not influence the network operation. A 'sniffer' node – one that continuously receives messages and never sends – can collect the contents of the join messages for further analysis. It is possible to send out this message at maximum strength, while sending the network messages at considerably lower strength[1]. In this way information of the whole network can be collected even when the nodes do not perceive the network as fully connected.

This is an attractive option because the infrastructure is already present and the data is gathered at a central point. It is available for analysis immediately after the experiment, allowing for quick feedback on the results. The drawbacks of over-the-air data collection are a limited bandwidth, and missing data due to collisions or interference. In the following part of this section, the feasibility of over-the-air data collection is investigated.

If every node chooses a broadcast slot for the join message randomly, we can approximate the probability that a message is successfully sent by the probability of not colliding (see section 3-5):

$$\Pr(\neg C) = (1 - \frac{1}{N_s})^d$$

For frame times of 1, 5 and 10 seconds, using the network settings that will be introduced in a moment, these numbers are displayed in Figure 8-4. The amount of data lost can be considerable, especially for short frame times and large networks.

The message loss could be prevented by assigning every node a unique slot to send its join message. This introduces a dangerous periodicity in the network and most likely all join-related functionality will be broken. Since we are not too self-assured about the performance of the proposed synchronization algorithms, and manually resetting hundreds of nodes is a lot of work, we will accept the limited data loss.

[1]This turned out to be theoretical, since a bug in MyriaCore prevented the join messages from being sent at full power.

Table 8-1: The contents of an experiment data logging message (for 8 active slots)

Item	Size [bytes]	Max occurrences per round
Identifying Token	1	1
Algorithm state	4	1
Temperature	2	1
Number of neighbours	1	1
Neighbour ID	2	7
Phase difference	2	7
Slot difference	1	7
Standard header	17	1
Total max	59	

For the large scale experiments, sniffers were positioned at strategic points to capture the data. This data was logged on laptops and later collected, merged and deduplicated into one logfile. Given the number of nodes that were heard and the number of rounds logged, we can calculate the the amount of messages expected. In Figure 8-3, this ratio is plotted. Some parts of the basement were used as storage, others were completely empty. Perhaps surprisingly, the empty rooms had the worst sniffer coverage. Overall, the sniffer coverage was well above 80% for many nodes. Only in the really large experiments (200+ nodes), some regions were seldomly heard and we will have to assume that the logged data is representative for those regions too. More sniffer coverage percentages can be found further on in Table 8-3.

8-1-3 Settings

For the experiments, the round times, size of the network, outside temperature and of course the algorithm used were varied. In this section the settings of the network that were not varied are listed.

Per node, the following information was captured per round:

- its neighbours heard in this round;
- the phase differences with those neighbours;
- its algorithm state;
- the temperature measured by the node.

This requires a message size of at least 59 bytes, see Table 8-1. A multiple of 16 is required by MyriaCore, so the message size was chose to be 64 bytes.

We want to take very broad guard times for our experiments, to make sure there are no messages lost due to timing errors. When something irregular like that happens, we want to see it. The guard times are listed in Table 8-2. Since the time a message of 64 bytes takes to transmit at 2 Mbps is about 15 ticks, these settings are clearly not advisable for any other application. The number of active slots every round was 8, and scheduling was allowed.

Table 8-2: The guard times chosen for different frame times

Frame time [s]	Guard time [ticks]	Equivalent drift [ppm]
1	6	183
2	9	137
5	21	128
10	41	125
20	80	122

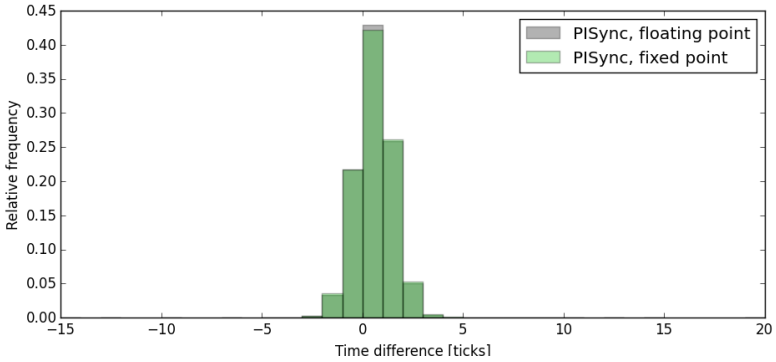


Figure 8-5: A floating point and fixed point implementation of the filter for PISync compared in simulation. The differences are extremely small.

8-2 Implementation

Moving algorithms from the simulator environment to embedded code required some adjustments to the algorithms, most notably the implementation of fixed-point filters. Since the processor on a MyriaNode does not possess a floating point processing unit, doing operations on decimal numbers is quite expensive in terms of time and energy consumption. The filters in MemoryMedian and PISync are of the following two forms:

$$\alpha_i(k+1) = (1-\rho)\alpha_i(k) + \rho \operatorname{Med}(\theta_i(k)) \qquad \text{MemoryMedian}$$
$$\alpha_i(k+1) = \kappa\alpha_i(k) + \epsilon_i \qquad \text{PISync}$$

Since ρ and κ are between 0 and 1, both involve decimal calculations. A way to circumvent these calculations is by implementing the filter in a fixed-point manner. The internal state of the filter is represented by a set of bytes, where the lowest bit does not represent 1 but a 2^{-k}th fraction of 1, where k determines the precision and range of the filter. Because the negative power of two, the filter state can be efficiently transformed back to the number in the correct units by bit-shifting.

There are two limitations to this implementation: the filter parameters have to be a negative power of 2, and the output of the filter will always be an integer, even when the state is between two integers. Since any correction will always be an integer too, this is a minor problem. To see if this change is allowable, the effect was evaluated in the simulator, see Figure 8-5. The differences are extremely small, as should be the case.

To be sure that the implementation functioned as intended, the algorithms were unit-tested: the outcome of sequences of time differences was compared with output from the simulator.

8-3 Results

The three series of experiments that were done, varying round times, network scale and temperature, will be presented in their own subsections here.

8-3-1 Round times

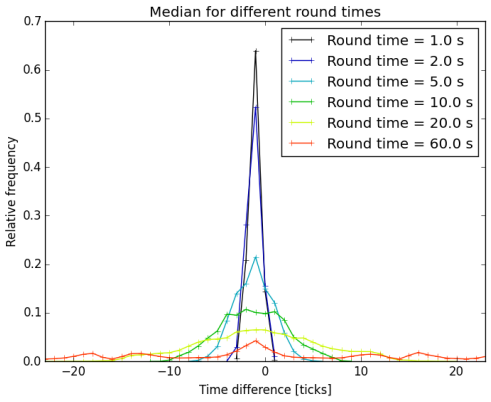
All three algorithms were run on small-scale, single-hop networks (11 nodes) for increasing round times. The results are shown in Figure 8-6, in the histogram form that was also used to assess simulation performance. All three algorithms show the same trend: for round times of 1 s and 2 s, synchronization is easy and all time differences are well between -5 and 5 ticks. All three algorithms have a roughly symmetric distribution of time differences, but also show a slight offset: negative time differences are more common than positive ones. This is the effect of quantization and misestimation of time differences: differences close to zero will be interpreted as negative.

Above the 2 s mark, performance starts to degrade in all cases. PISync and MemoryMedian hold up much better than Median however, an effect which is especially evident for the 60 s experiments. In Figure 8-7, the standard deviations and maximum of each algorithm per roundtime is plotted. Median is linear in both measures, as can be expected from a proportional algorithm. Both drift-estimating algorithms seem fairly linear as well, but with a smaller angle. PISync struggles a little with very small round times, and has the worst maximum errors there. It is the best performer in terms of maximum errors for the highest two round times, and otherwise competes with MemoryMedian for the lowest standard deviation. This is positively surprising, because the tuning of the algorithm was not changed (apart from e_{max}), even though this was required in its original form.

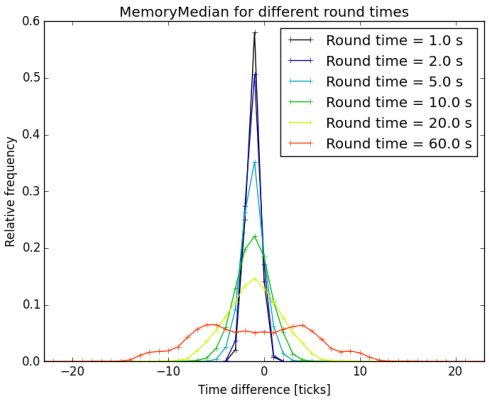
In Figure 8-8, the minimum, maximum and average time differences per round are plotted for the three algorithms with the 20 s round times. The improved performance of MemoryMedian and PISync is once again clear. In MemoryMedian, one can see the algorithm converging on the first couple of rounds, distorting the maximum a bit. PISync does not have this transition period, but there are some unexpected outliers in the minimum and maximum.

The filter behaviour of both algorithms is shown in Figure 8-9. MemoryMedian has smooth transitions, and the algorithm states remain very steady. Due to the quantization of corrections, the filters are attracted by integers. If a node measures similar time differences over time, the filter will increase in value until it has reached an integer value and the time differences become smaller. The filter values of PISync are more variable. Sudden increases in the value are intermixed with slow decreases in between; characteristic of the filter structure chosen. Lines with the same color belong to identical nodes, and one can see that there is some consistency in the drift estimates: both algorithms have node 3 and 4 approach negative values and 2 and 6 positive.

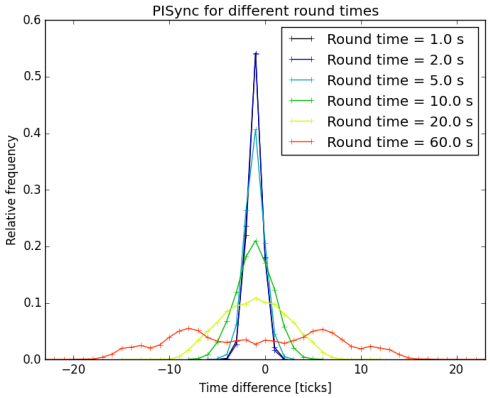
A remarkable result is shown in Figure 8-10, the histogram of time differences for Median at 60s round time. This experiment was done with only six nodes. At first the histogram seems wrong; it is completely different from the gaussian-like histograms we have seen so far. A possible explanation could be that this network is so small that in the absence of drift correction there is only a limited number of time differences that is possible. A node has a fixed ratio of drifts with all its neighbours. It will accumulate the same time difference with each of these neighbours every round, and will pick one of these neighbours as a median. If there are only five neighbours, there are a limited number of corrections that can be done. The stochastic parts of this process are not sufficient to satisfy the central limit theorem, leading to these large fluctuations in likelihood of errors. If the



(a) Median



(b) MemoryMedian



(c) PISync

Figure 8-6: The histograms of time differences for all algorithms, for different round times.

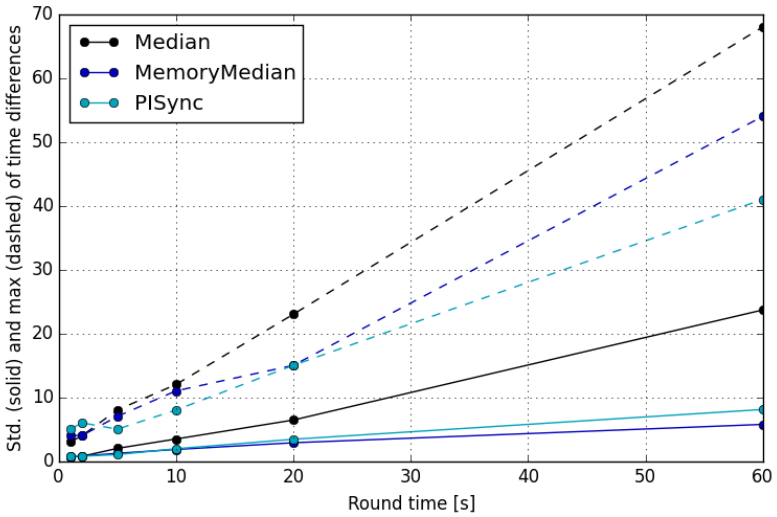
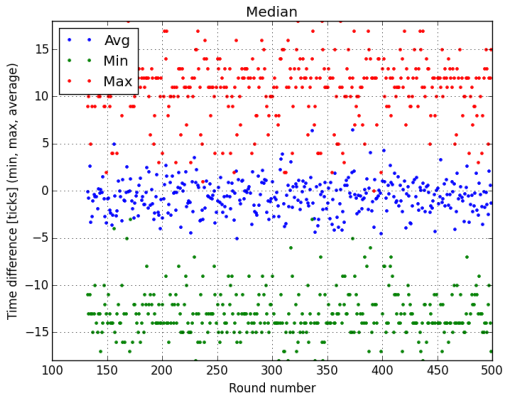
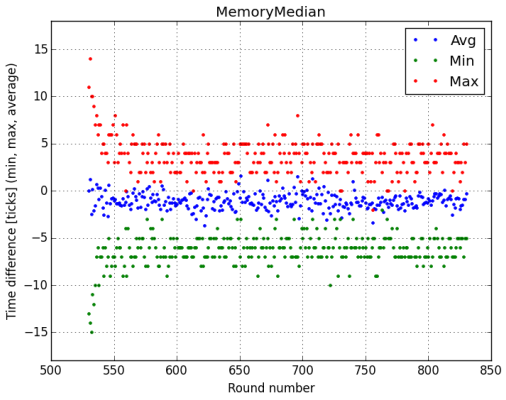


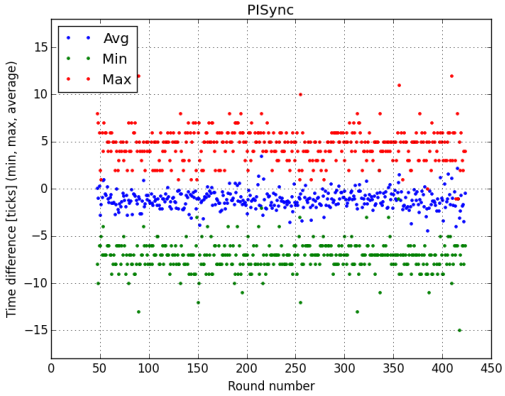
Figure 8-7: The standard deviation and maximum of time differences compared between the algorithms for different round times.



(a) Median

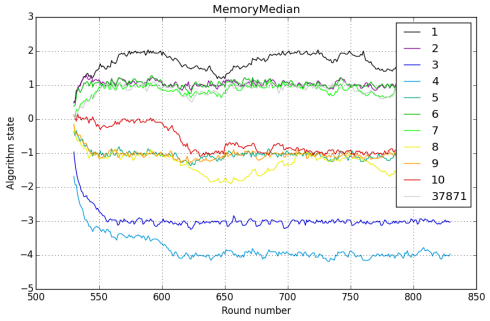


(b) MemoryMedian

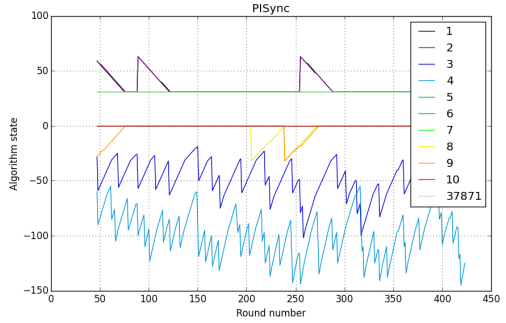


(c) PISync

Figure 8-8: The time differences over the 20 second round time experiments.



(a) MemoryMedian



(b) PISync

Figure 8-9: The algorithm states over the 20 second round time experiments. Since the states use very different representations, their values can not be directly compared.

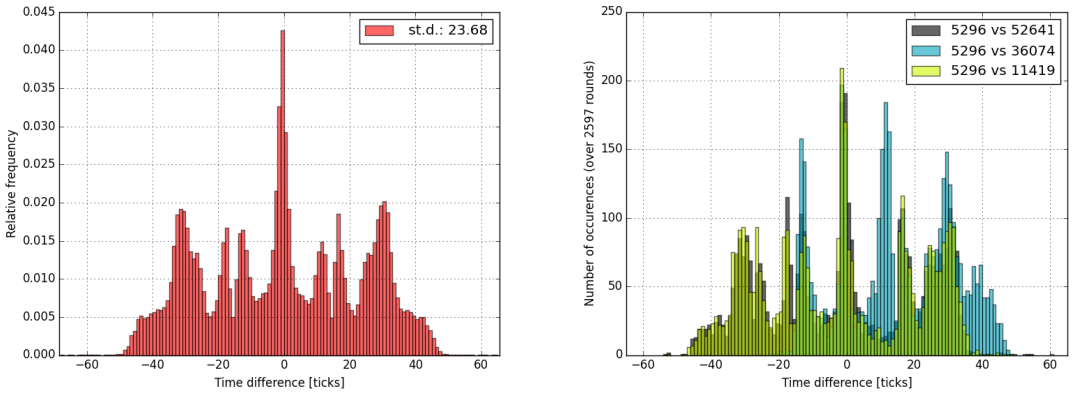


Figure 8-10: The histogram of time differences for Median at 60 seconds, for all nodes (a) and between three pairs (b).

time differences between pairs of nodes are plotted (the second graph in Figure 8-10), this theory is confirmed: some extremes are unique to certain pairs of nodes. It is only at a 60 second round time that the granularity cannot mask this effect.

8-3-2 Scale

The series of experiments with increasing scale was conducted on a network running at a frame time of 10 s. The connections in the largest network deployment are shown in Figure 8-11. The network is well connected, and counting all connections the maximum distance is 4 hops. This involves some low-probability links however, so the diameter is perhaps higher. Note that the connection ratios were not corrected for sniffer coverage, so connections between two nodes that were seldomly sniffed are underrepresented. The central board clearly functions as a hub in the network.

In Figure 8-12 the histograms for all three algorithms are shown. Median and MemoryMedian are hardly influenced by network size, and there is no pattern in the variability that remains. PISync on the other hand breaks down for the network sizes above 113 nodes! The histogram maxmimum is much lower for the experiments of 190 and 285 logged nodes, and it is much wider. There is still a small majority of near-zero time differences, but there are also many time differences of hundreds of ticks (still remaining in the active period). It seems like the network size has an influence on the local synchronization in this case.

To investigate the matter further, the logs of the leftmost sniffer are plotted separately in Figure 8-13. This sniffer is situated in a room that is well-connected, and has a few links to nodes outside the room. This log is one of the more interesting ones, because the instability is only intermittently present here. In the time differences over time, there are clearly moments in which the network is well-synchronized, and larger periods in which this equilibrium is disturbed. These effects correlate with the drift estimations of the nodes: time difference lead to large drift estimations, leading to even larger time differences.

In the third figure, which is a transpose of the plot in Figure 7-24 zoomed in on the active period, this process is more visible. During the instable periods, the algorithm struggles to keep the nodes together, presumably because some of them are ‘dragged away’ by nodes that were not heard by the sniffer. At a certain moment around round 17100, the conflicting nodes break off the rest of the group, actually improving the synchronization, but separating this part of the network in three parts. This situation stays intact, with some disturbances along the way. By the end of the experiment, the original group has

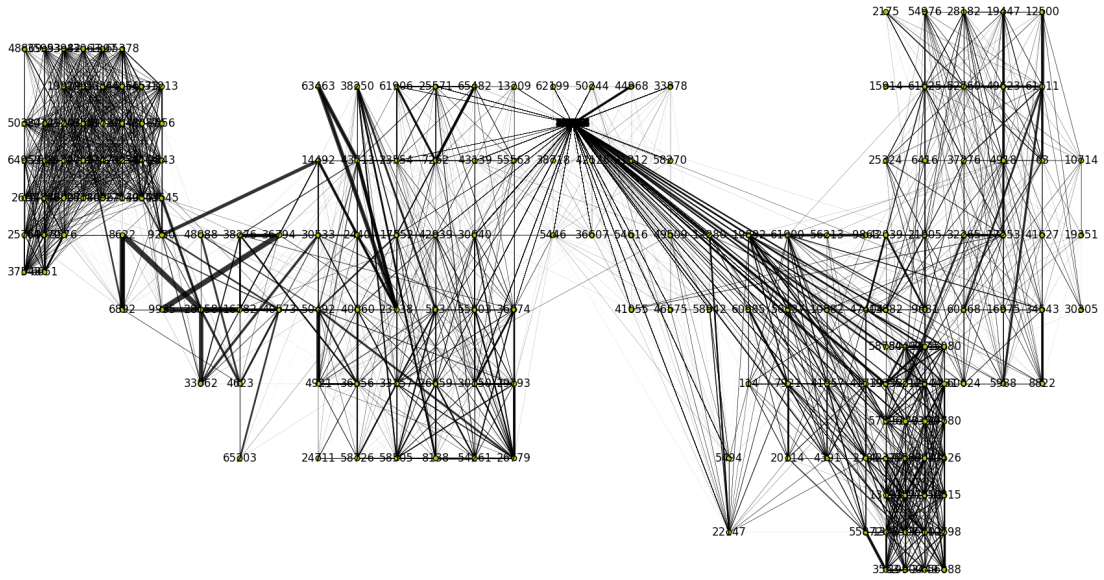


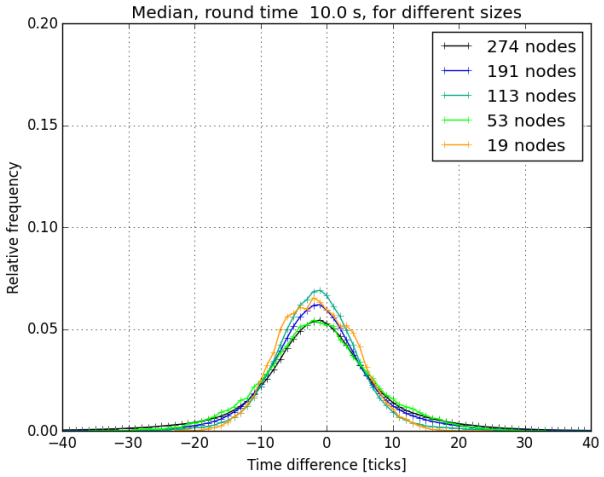
Figure 8-11: The same map as Figure 8-3, with the connections indicated. The thickness of lines is a measure for the average reception ratio over an experiment.

grown quite small and the number of schedules is decreased. This leads to a shorter active period.

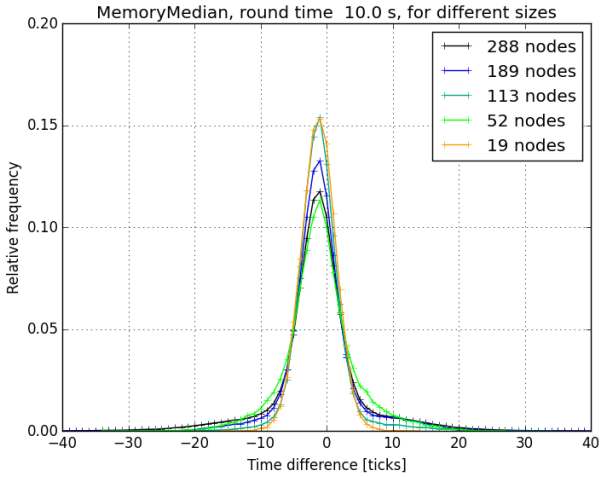
From these figures, we can conclude that there are groups of nodes concurring on a common drift, but that these groups are conflicting. The network splits up into regions that disagree. When looking at individual node data, there is significant difference in the average time difference with neighbours and the standard deviation of time differences, see Figure 8-14. The nodes on the center board, where node density is extremely high, have the largest and most variable time differences. Although the differences in the rest of the network are smaller, they are still very large when compared to a good synchronization. When the differences and variance for a well-performing algorithm are plotted on the same scale, the circles would not be visible.

The last figure for PISync only solidifies our understanding of the problem. In Figure 8-15 the average absolute time difference per node is plotted against the number of unique neighbours. There is a strong correlation: nodes from the very dense neighbourhood have extreme time differences, and nodes from more sparse neighbourhoods are increasingly influenced by this. As long as the critical density of about 100 nodes is not crossed, the algorithm remains stable.

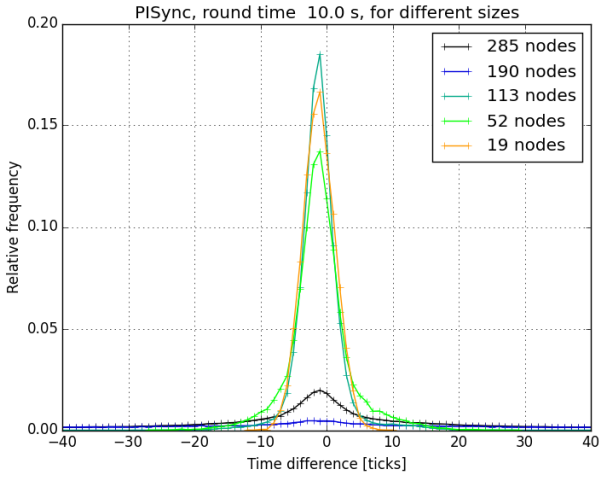
The fact that MemoryMedian does not suffer from this effect suggests that it is not just the drift estimation that is hampered by extreme density, but perhaps also using the average for proportional correction is risky. In any multihop network, time differences are likely to increase over several hops. Taking incidental long-range (and thus large-error) messages into account for the synchronization might reduce these differences, but could also be risky: too many outliers will only introduce instability in the network, as happened with PISync. Another possible cause could be that PISync is tuned too aggressively, and should be more conservative in estimating drifts.



(a) Median



(b) MemoryMedian



(c) PISync

Figure 8-12: The histograms of time differences for all algorithms, for different network sizes.

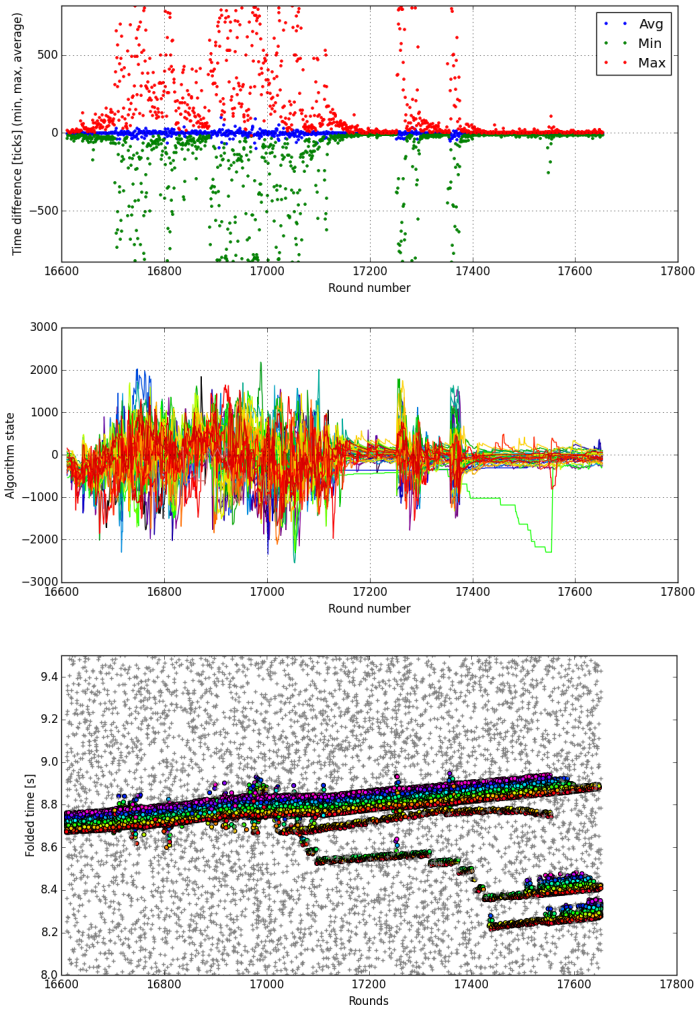


Figure 8-13: The time differences, states and timing information measured by one sniffer in the 300+ node PISync network. In the last figure, colored dots are messages from the active period and gray crosses are join messages. The moment of reception by the sniffer in the 10 second rounds is folded on the y-axis, thus showing the timing of different nodes.

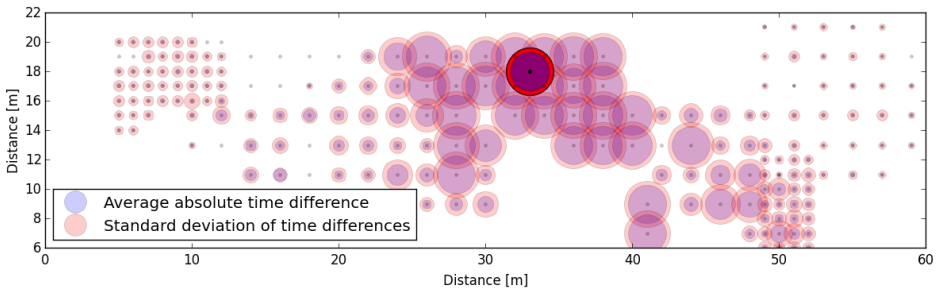


Figure 8-14: The average of absolute time differences and the standard deviation of time differences measured by every node of the 300 node experiment running PISync. The sizes of circles in the legend correspond to 200 ticks.

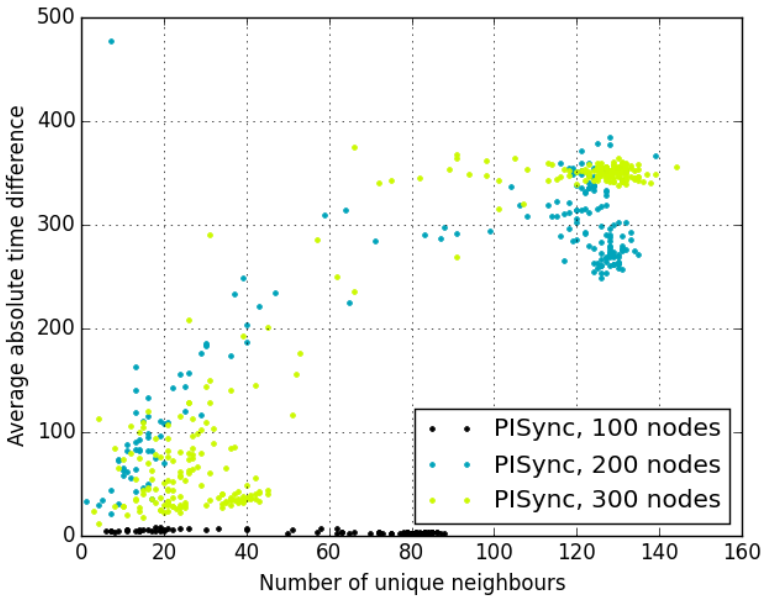


Figure 8-15: The average of absolute time differences measured by every node, plotted against the number of unique neighbours over the whole experiment.

8-3-3 Temperature Compensation

As was concluded in section 4-3, the rudimentary temperature sensor that is integrated in the hardware platform might be used for a feedforward temperature compensation. Experiments were done to see whether this works in practice too.

Validation of Internal Temperature Sensor

Apart from having a terrible accuracy, the internal temperature sensor is riddled with anomalies. There are 5 different Product Anomaly Notices (PANs) [44] for this component that luckily can be worked around with software. Nevertheless, it's no wasted effort to validate the internal temperature sensor ourselves. We used MyriaNed nodes equipped with a TMP112 [68], which has an accuracy of $\pm$ 0.5 °C, much better than the $\pm$ 4 °C (or even $\pm$ 8 °C in extreme cases) of the nRF51. We ran experiments with multiple sensors, doing measurements of both sensors simultaneously.

In Figure 8-16, the result of the experiment is shown. The internal temperature sensor is quite precise: there is very little noise and it detects very subtle changes in temperature. It has a very large offset however. The offset is usually around 13 °C, but one node has an offset of about 10 °C. When the offset is subtracted, the variability in between nodes is within the $\pm$4 °C as expected.

This offset of about 13 °C is not expected, but does not vary much over environmental temperatures. This data has been communicated to Nordic, but no satisfactory explanation was received. To circumvent the issue, a rudimentary calibration was done were all nodes determine their offset at startup, which has to be done at a known temperature.

Synchronization Improvement

The results of the experiment done using the climate chamber are shown in Figure 8-17. The temperature inside the chamber was quickly raised to 65 °C after the start of

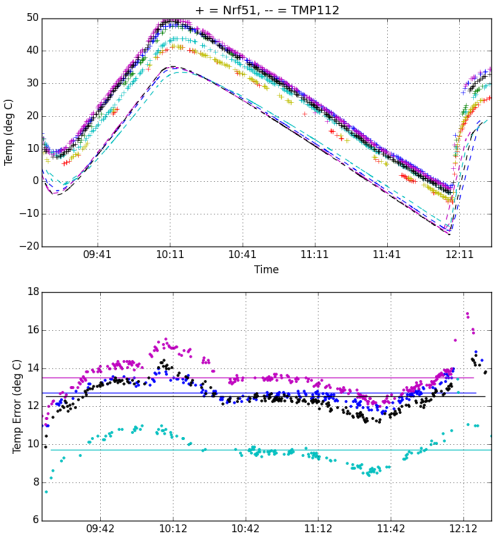


Figure 8-16: A temperature experiment with eight nodes. Half of the nodes has a TMP112 sensor (dashed lines) in addition to the internal sensor (crosses) on every node. The second graph shows the difference between both sensors, and the average difference.

the experiment. Over the course of several hours (the round time is 15 seconds) the temperature was decreased in steps of 10 °C until −25 °C. Finally the temperature was raised suddenly towards 65 °C again and then the door of the climate chamber was opened to have natural cooling.

The results are very convincing. With just the Median algorithm, the temperature dependency shows nicely, with the time differences even following an approximately quadratic curve for linear increases in temperature. Adding a temperature compensation tempers the time differences and increases the operational range for a given guard time. Surprisingly, using MemoryMedian without temperature compensation works better than Median with temperature compensation. The adaptive drift estimation can better compensate for persistent time differences than the feedforward temperature compensation, which is necessarily inaccurate. Indeed, when looking at Figure 8-18, MemoryMedian's state estimations (without temperature compensation) follow the temperature differences. Combining both methods yields best results, and with that method there is almost no effect visible in a range from 50 °C to −15 °C.

8-4 Comparison

To see whether we could have predicted the results of the experiments with our models, a comparison is made in this section between the simulations and the experiments. To avoid the complications of a radio model (which is a field of study in itself), we will use the topological information from the experiments and feed them into our simulations. This information is not complete however, as is inherent with the over-the-air data retrieval method. In Table 8-3 an impression of the coverages for experiments is given as the ratio between successfully received messages and the expected number of messages based on the number of nodes heard and the number of rounds of the experiment. As expected, the coverage becomes lower with increasing network size, especially around 300 nodes.

One unknown parameter remains, and that is the drift per node. The temperature of the experimental setup was not far from room temperature. A reasonable assumption is therefore that the drifts of the nodes are uniformly drawn from a range. To 'calibrate' the

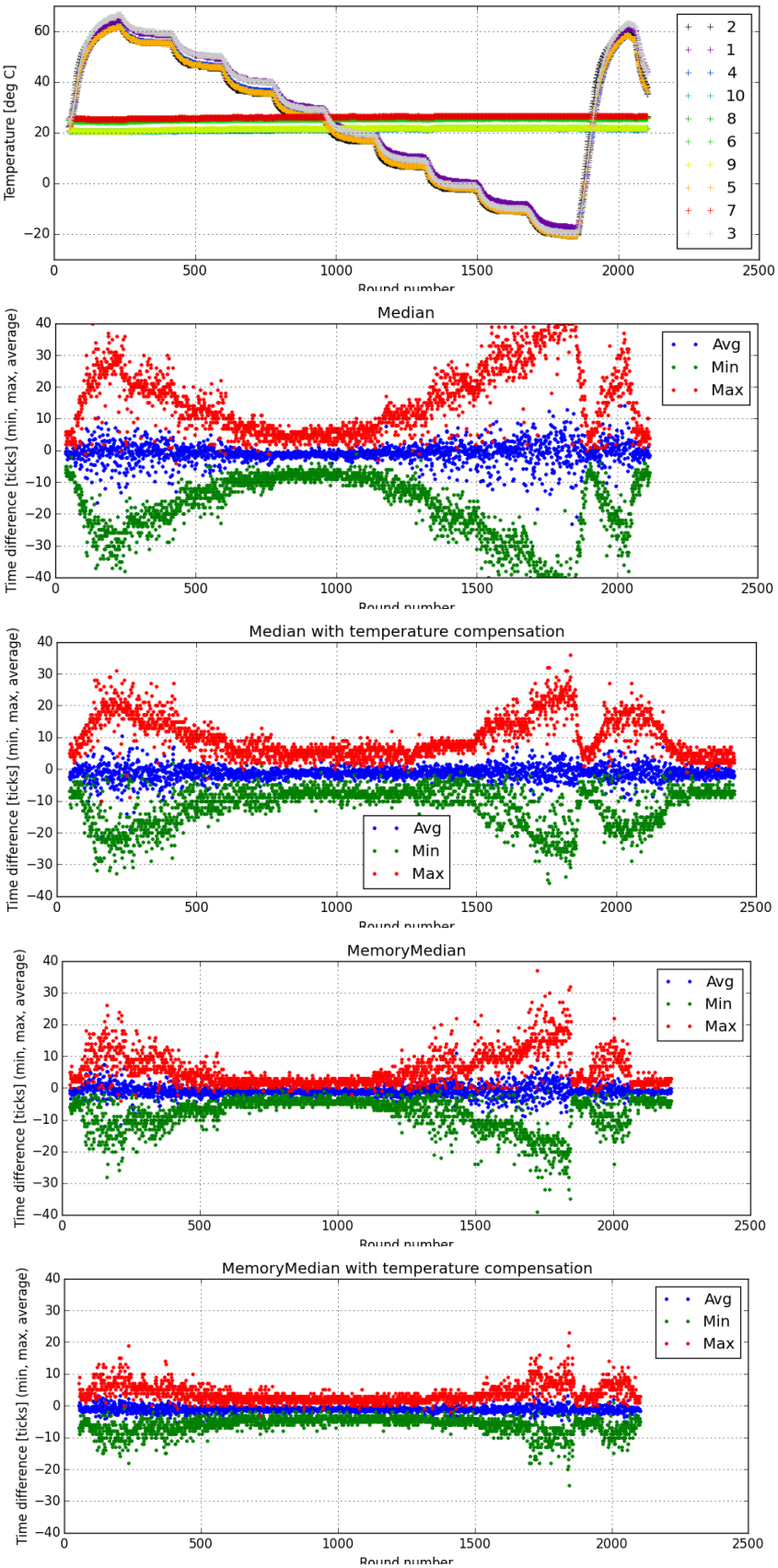


Figure 8-17: The performance of Median and MemoryMedian on a 10 node network at 15 seconds round time, under extreme temperature differences, with and without temperature compensation. The temperature in the first graph was measured with the internal sensor.

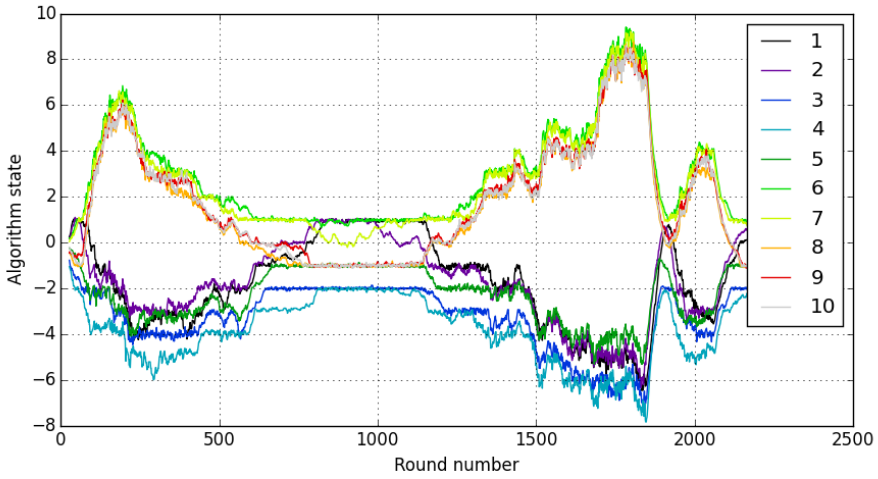


Figure 8-18: The algorithm state of MemoryMedian in the uncorrected temperature experiment.

Table 8-3: The data coverage of the experiments. Since some nodes were never heard, this is an overestimation.

Experiments	Coverage		
	min	mean	max
Round times (11 nodes)	88.65%	96.02%	99.74%
20 nodes	86.82%	90.52%	92.74%
50 nodes	87.51%	90.42%	92.19%
100 nodes	88.72%	90.36%	91.99%
200 nodes	89.41%	90.63%	91.73%
300 nodes	68.79%	74.45%	77.83%

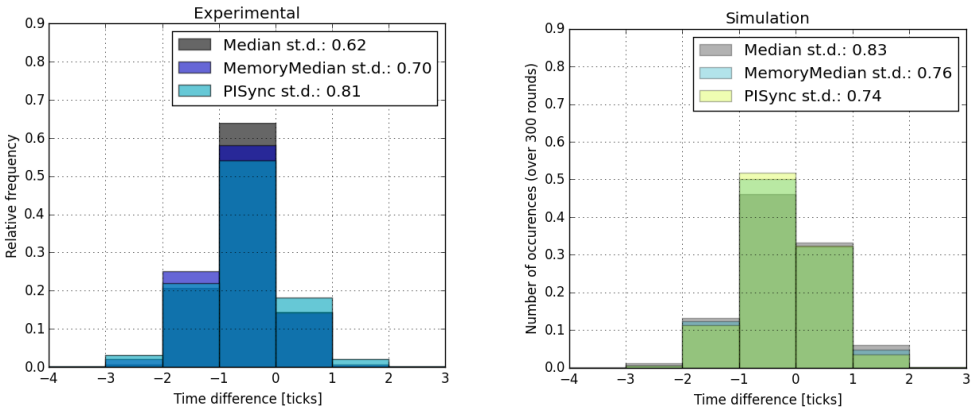


Figure 8-19: The histograms of time differences from simulation and experiment, both with 11 nodes and a frame time of 1 second.

Table 8-4: The maximum simulated time difference minus the maximum measured time difference for round time experiments

Round time [s]	1	2	5	10	20	60
Median	4	0	-2	5	-1	-17
MemoryMedian	-2	-2	-3	-4	-1	-14
PISync	-1	-4	-2	-4	-4	10

range to use in our simulations, the simple 11 node, 1 second experiments were compared. In Figure 8-19, the histograms are shown after tuning the range of drifts in the simulator. A range of ±8 ppm was found to fit the results best. This is smaller than the ±20 ppm that might be expected from the data sheets, but that is an upper bound of course. The performance of Median is a bit better than predicted, and PISync a bit worse. In both simulation and experiments, the differences between the algorithms are small.

Using this distribution of drifts, the simulator was used to reproduce the synchronization process on networks with increasing round times. In Figure 8-20, the same graph as Figure 8-7 is displayed. The graphs look similar: all algorithms have an approximately linear dependency on frame time in both the standard deviation of time differences. Only the standard deviation of PISync on 60 s is not as expected, and higher than in reality. PISync and MemoryMedian are generally better than Median, whose performance is often worse in simulations than in reality. If we disregard the unusually high frame time of 60 s, the predictions of maxima by the simulations are quite good: they are mostly within two to four ticks of the actually measured maximum, see Table 8-4. There is no guarantee that this result is consistent however, since the distribution of drifts is random in the simulator.

The simulation results for different network sizes are displayed in Figure 8-21. Those plots should be compared to Figure 8-12. Due to the scaling this is not visible, but the extremes of time differences for the simulations are much larger. This is probably caused by limited sniffer coverage: some nodes are heard just a few times, and thus will have very few communication instants in the simulation. In between this communications the drift can accumulate without correction.

Aside from these extremes, all three algorithms perform better in simulation than in reality: the variability in differences is lower, and the frequency of near-zero errors is much higher. The differences in performance between different scales are quite large in the simulations, but there seems to be no trend: 19 nodes perform well, 53 nodes is terrible, 113 nodes is among the best performances again. It's striking that the order from best to

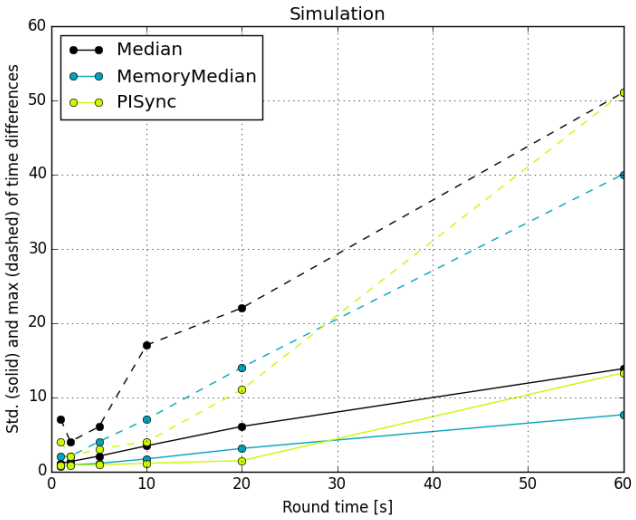


Figure 8-20: The simulator's results for the algorithms on different round times, using topology information from the experiments with 11 nodes. This figure can be compared to Figure 8-7.

worst (19, 113, 190, 285, 52) is almost the same for all three algorithms. This indicates a common influence on performance, perhaps the communication topology as it is logged.

Unfortunately, the breakdown of PISync is not predicted by the simulations, even though the network density is almost the same. Due to limited sniffer coverage, the simulation cannot be done under the exact same circumstances. The performance of the simulation of 285 nodes is the worst, but nowhere as bad as the experimental one. In any case, this discrepancy between the simulations and reality is an interesting problem that prompts further research. Unfortunately it is out of the scope of this MSc project, which can be thought of as a first iteration.

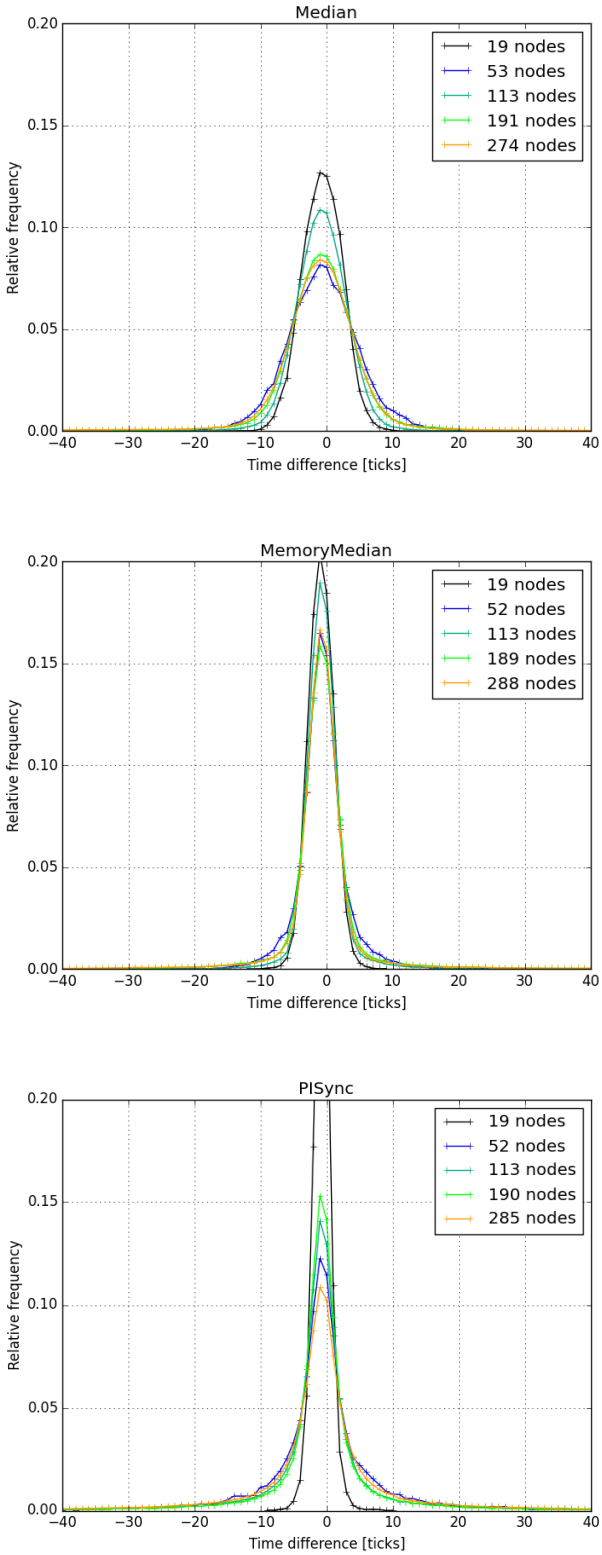


Figure 8-21: The histograms of time differences generated by the simulator on different network sizes, for the three algorithms, at a round time of 10 seconds.

Chapter 9

Conclusions

The goal of this MSc project has been to develop a better synchronization algorithm for an existing MAC protocol (gMAC) that is used in a commercial WSN called MyriaNed. An improved synchronization has the potential to dramatically increase energy efficiency. We have first explored the problem and established the solution space in chapters 2 and 3. It has been shown that if the synchronization in gMAC is insufficient (requiring guard times that are more than half the transmission time of a message), it is a rather inefficient protocol, and there are most likely better alternatives.

There is a need for specialized algorithms due to the unique challenges of wireless mesh networks, and the intimate coupling with the MAC protocol. In chapter 4 an overview of algorithms available in literature is presented. Most algorithms use a form of clock drift estimation, and this technique is likely to enable improvement upon the current synchronization algorithm (Median). We propose three candidates: two adapted from published algorithms (ATS and PISync) and one inspired by Median (MemoryMedian). The performance of these algorithms has been evaluated in simulations and experimentally with networks of up to 302 nodes.

Average Time Sync (ATS) uses two cascaded consensus protocols to establish the drift and offset of a network-wide virtual reference clock that all nodes try to follow. In simulations this algorithm shows the best drift estimation properties, but its performance is volatile: both the best and the worst results are produced by this algorithm. It is not robust against sudden errors, and is the only algorithm that requires information to be included in messages. For these reasons it was not considered for the experimental evaluation.

PISync implements a PI controller for every node. A correction of the clock consists of the average of the measured time differences, and a value that was integrated over time. This integration is done using some heuristics to prevent integrating errors that can not be caused by drift. Its performance looks extremely promising in simulation, but suffers from a speedup due to nonzero-mean error integration, a form of integrator windup. To circumvent this, a filter is introduced that sacrifices some performance for improved reliability. Experiments have shown however that extreme node density leads to instability in this algorithm. Our simulations can not reproduce this instability, and thus the mechanism from which it arises remains largely unclear. Aside from the instability of PISync, network size has no influence on synchronization performance of the tested algorithms.

MemoryMedian builds on the existing synchronization algorithm by taking the median of measured time differences to apply a correction every round. The median value is fed

through a simple low pass filter as well, providing an additional correction that is a form of drift compensation. The performance of MemoryMedian is better than that of Median in simulations and experiments alike. MemoryMedian proves to be more robust than PISync and ATS, since it does not suffer from problems with network density. It provides a steady improvement over Median in all cases.

In addition, a feed forward temperature compensation rule based on available low-quality temperature measurements has been shown to reduce the influence of temperature considerably. This feed forward rule works well in combination with MemoryMedian. Using both temperature compensation and MemoryMedian increases the temperature range that can be covered without additional guard times from 15 °C to 35 °C to −5 °C to 45 °C.

An important part of this thesis has focused on providing analytical insight into the dynamics of synchronization processes on networks like MyriaNed. We have been able to model a class of synchronization algorithms (first order algorithms) as a consensus process. For this class of algorithms it can be proven that the synchronization process converges if the union of links in the network forms a spanning tree often enough – a condition that is satisfied by MyriaNed.

In an attempt to extend these results into proofs of stability for drift estimating algorithms (second order algorithms) we have constructed a framework in which Median, MemoryMedian and PISync can be described. Unfortunately, the techniques used for first order algorithms can not readily be applied to this framework.

9-1 Recommendations & Future Work

Based on the results of this thesis we would recommend Chess Wise to work on incorporating MemoryMedian in MyriaCore. There are still some aspects to be addressed. The tuning of the gains of the algorithm was only done in simulations, and perhaps there are more optimal values. This should be investigated by doing a series of experiments with varying gains. Secondly, the algorithm was not tested on dynamic networks, where the neighbours change frequently. Since the drift estimation has to change accordingly, this could lead to degraded performance. On the other hand, having more neighbour diversity could enable nodes to be less biased in their estimation.

In addition, using the temperature compensation technique has clear benefits and works both with Median and MemoryMedian. The implementation could be made more robust (by rejecting impossible values or changes) and a solution has to be found for the large offset of the internal temperature sensor.

When these algorithms are incorporated, the guard times could be reduced to the values in Table 9-1. Note that this table suggests that even with only Median, the default guard times (9 ticks) are not needed in most situations.

In our experiments, we have always made sure that guard times were more than sufficient. Experience shows that throughput does not gradually lessen with decreasing guard times. At a certain point, the shortage of guard time leads to so many collisions that only very few messages remain. These messages are not sufficient for nodes to correct their clock adequately, and the next round the time differences are even larger, leading to even more collisions. This point of breakdown could be investigated experimentally. Ideally one would like to predict such a point, but our models do not incorporate this effect. Perhaps theory on crystallization or epidemics on networks can provide insight.

Another important aspect in which our models fall short is the failure to reproduce PISync's breakdown in high-density regions. Finding the mechanism for this failure,

Table 9-1: Recommended guard times, in ticks, for various round times, at room temperature.

Round time	Median	MemoryMedian
1 s	4	4
2 s	4	4
5 s	7	5
10 s	12	7
15 s	14	8
20 s	16	9
60 s	50	14

whether it is the averaging action, tuning or something else, can yield valuable insights in the influence of network communication dynamics on synchronization.

Further improvements that could be worthwhile to investigate are:

- Adaptive guard times: the guard times are currently static, and set for the worst case. This might not always be necessary. If mechanisms can be found to reliably predict time differences, guard times might adapt to circumstances, increasing efficiency even further.
- Multirate networks: in some infrastructures, it could be beneficial to have different round times in one network. This would influence synchronization, and can be both beneficial (perhaps the nodes on longer round times can use the shorter round times to attain sync before communicating) or detrimental (the shorter round nodes have to accommodate for the longer rounds with their guard times).
- Network slowdown: in experiments, MemoryMedian showed a transient at the start in which filter values had yet to settle. For very long round times this can be troublesome, since the initial errors might lead to less throughput, and the network might break before the estimations had a chance to attain a good value. To get networks to run at very long round times with small guard times, a scheme can be devised where the nodes start communicating in short rounds, and once the estimates are settled transition to very long rounds.

Finally, continuing the analysis of chapter 5 and finding conditions for convergence of second-order algorithms on directed, switching topologies in general could be a valuable result for far more applications besides synchronization. It would mean a serious extension to the already valuable theory on consensus algorithms. Another possible extension to this theory would be to find more conservative error bounds for consensus processes, or bounds for convergence on advanced topologies.

Appendix A

Theorems and Definitions

A-1 Theorems

A-1-1 Geršgorin Circle Theorem

The Geršgorin Circle Theorem provides bounds on the regions in which eigenvalues of square matrices may lie. It finds a special application in determining the eigenvalues of Laplacian matrices. The following is reproduced from [51].

Every eigenvalue λ of an $m \times m$ matrix $A = [a_{ij}]$ lies in at least one of the discs

$$|\lambda - a_{ii}| \leq P_i = \sum_{j \neq i} |a_{ij}|, \quad i = 1, 2, \ldots, m \tag{A-1}$$

That is, every eigenvalue lies in at least one of the n discs in the complex plane with centers $|a_{ii}|$ and radii equal to the row sums of off-diagonal terms.

A-1-2 Wolfowitz's Lemma

Let $M_1, M_2, \ldots, M_m$ be a finite set of ergodic matrices with the property that for each sequence $M_{i_1}, M_{i_2}, \ldots, M_{i_j}$ the product $M_{i_j} M_{i_{j-1}} \ldots M_{i_1}$ is ergodic too. Then for each infinite sequence of those sequences:

$$\lim_{j \to \infty} M_{i_j} M_{i_{j-1}} \ldots M_{i_1} = \mathbf{1}c$$

Where c is a row vector.

A-1-3 Lyapunov Theorem

Lyapunov theory can be used to establish stability properties of dynamical systems. This formulation of the theorem is taken from the course slides of SC4025 Control Theory, TU Delft.

Consider a dynamical system $\dot{x} = f(x)$ with $f : \mathcal{D} \to \mathbb{R}$ defined on $\mathcal{D} \subset \mathbb{R}^n$ and an equilibrium point $f(x_e) = 0$. A function $V(x)$ is a Lyapunov function if $\frac{d}{dt}V(x(t)) \leq 0$ (which implies $[\delta_x V(x)]\, f(x) \leq 0$ via the chain rule), and $V(x_e) = 0$. A Lyapunov function can be seen as a 'potential function' of state trajectories. If this function is monotonically decreasing, the states will converge to the lowest potential: the equilibrium. Formally:

- If $V(x) > V(x_e)$ for all $x \in \mathcal{D}\backslash\{x_e\}$, then x_e is stable (the state trajectories stay bounded).
- If $V(x) > V(x_e)$ and $[\delta_x V(x)]\, f(x) < 0$ for all $x \in \mathcal{D}\backslash\{x_e\}$, then x_e is asymptotically stable (the state trajectories converge).

So loosely speaking, if a positive definite function $V(x)$ can be found that is monotonically non-increasing for all trajectories of the dynamical system $f(x)$, stability is proven. If the function is monotonically decreasing for all trajectories, asymptotic stability is proven.

A-2 Definitions

Definition A.1 (Stochastic Matrix)**.** *A matrix A is stochastic if and only if (i) its entries are nonnegative, (ii) its row sums are 1.*

The product of two stochastic matrices is again stochastic.

Definition A.2 (Primitive Matrix)**.** *A square, nonnegative matrix A is called primitive if A^m is all-positive for m sufficiently large.*

A sufficient condition is for the matrix to be a nonnegative, irreducible matrix with a positive element on the main diagonal.

Definition A.3 (Ergodic/SIA Matrix)**.** *A square, nonnegative matrix A is called ergodic if the rank of $\lim_{i\to\infty} A^i$ is 1. Ergodic matrices are also called stochastic, indecomposable, aperiodic (SIA) matrices.*

Stochastic, primitive matrices are ergodic [32].

Definition A.4 (Irreducible Matrix)**.** *A matrix A is irreducible if there is no permutation $P^T AP$ that makes it upper block triangular. The adjacency matrix A corresponding to a directed graph is irreducible if the graph is strongly connected.*

Definition A.5 (Uniform Boundedness)**.** *A group of values or functions is uniformly bounded if all values or functions are bounded, and there is one bound within which all of them fall.*

Definition A.6 (Balanced graph)**.** *A graph where every node has as many incoming as outgoing links. An undirected graph is balanced by definition.*

Definition A.7 (Convexity)**.** *Any value e lies within the convex hull of values $\{x_1, \ldots, x_n\}$ if e can be expressed as:*

$$e = \sum_{i=0}^{n} \lambda_i x_i$$

where:

$$\lambda_i \in [0, 1]$$

$$\sum_{i=0}^{n} \lambda_i = 1$$

In the case of x_i and e being scalars, e lies within the strictly convex hull of $\{x_1, \ldots, x_n\}$ if it's in between (and not equal to) $\min(\{x_i\})$ or $\max(\{x_i\})$.

Definition A.8 (Spanning Tree)**.** *A graph in which there exists a path from at least one node to all other nodes is said to contain a spanning tree. The spanning tree is the minimal subgraph (containing all nodes) for which the property holds.*

Appendix B

Additional gMAC Statistics

In section 3-5, the transmission probabilities of gossip MAC (gMAC) in a single neighbourhood were analyzed. This line of thought can be continued by looking at the success rate over multiple attempts, the influence of a scheduling strategy and trying to make predictions of expected message reception ratios. This has little to do with synchronization (since it all assumes a well-synchronized network), but provides insight in the dynamics of MyriaNed.

Note: in this appendix d_i is used for the number of neighbours instead of m_i.

B-1 Multiple attempts

The probabilities of a single transmission succeeding for reasonable choices of parameters are not very encouraging, see Figure 3-6. For a realistic ratio ($N_s = 8, d_i = 10$) the probability of transmission is around 30%, slowly decreasing with increasing node density. Luckily a node can attempt to transmit in multiple consecutive rounds, increasing the odds. A straightforward way to model this is viewing it as a binomial process. In every round the node has a transmission success of $p = (1 - \frac{1}{N_s})^{d_i}$, as modelled in section 3-5. Assuming that the chances of success are independent over n_r successive rounds, the probability of getting exactly k successful transmissions is

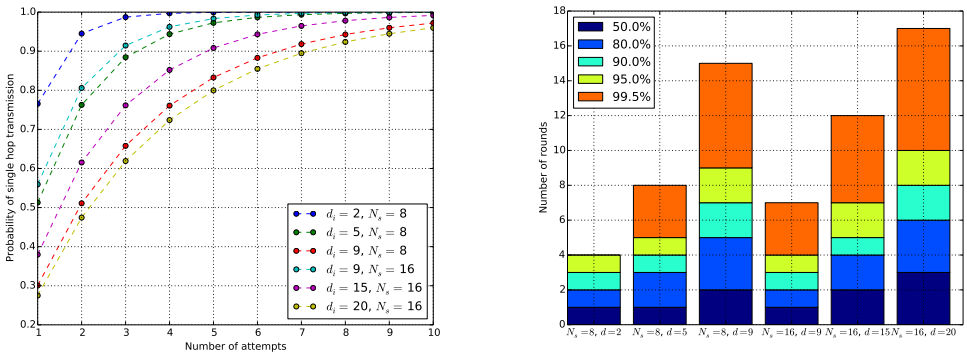
$$\Pr(X = k) = \binom{n_r}{k} p^k (1-p)^{n_r - k}$$

where X is the number of messages successfully transmitted. In order to determine the chances of success, it suffices to have *at least* $k = 1$ successful transmissions over all the attempts.

$$\Pr(X \geq k) = \sum_{i=k}^{n_r} \binom{n_r}{i} p^i (1-p)^{n_r - i} \tag{B-1}$$

Note that this is the 'inverse' of the cumulative distribution function of the binomial distribution, where the sum runs from 0 to k.

To our relief, the probabilities of success quickly approach 1 after several rounds (Figure B-1). As long as the number of slots for N_s is larger then the number of neighbours, the network does not hamper transmission too much.



(a) **(b)**

Figure B-1: The probability of a successful transmission over one hop over multiple rounds, in different settings for N_s, d_i, and the number of rounds needed for a specific confidence of success, for the same settings.

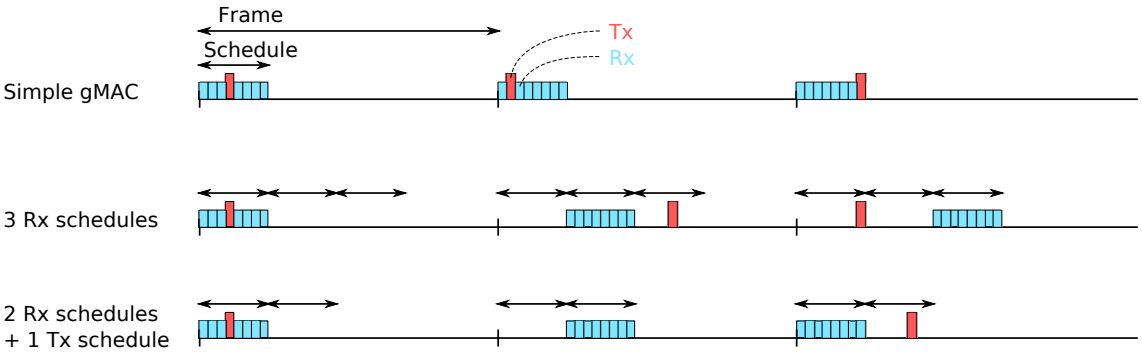


Figure B-2: Scheduling in MyriaCore.

B-1-1 Strategy

When a network (or a neighbourhood in a network) is very dense, the number of neighbours might exceed the amount of active slots. The probability of successful transmission becomes very low, and much energy is wasted. This issue is known, and countermeasures are taken. An extended version of gMAC is used, that employs a strategy for shifting the transmit and receive slots, effectively broadening the active period. This is called 'scheduling', and has its basis in the MSc thesis by Anemaet [3].

Two types of scheduling are done: Rx scheduling and Tx scheduling, see Figure B-2. Each node keeps a list of recently-heard neighbours to estimate its local density, and based on this neighbour count, schedules are increased or decreased.

Rx Scheduling If the neighbour count is larger the number of active slots minus 1, the number of Rx schedules is increased (up to a prespecified maximum). The active period of a frame is lengthened by one set of N_s slots, but the number of Rx and Tx slots is kept the same, to keep the energy usage predictable. In every frame, the Rx slots are shifted one schedule forward. The Tx slot is chosen randomly from all active slots, and may or may not coincide with an Rx slot.

Tx Scheduling The number of Rx schedules is limited (usually 3). If this limit is reached, and the number of neighbours is still larger than the active slots (the number of Rx

schedules times the number of receive slots) minus 1, a node tries to reduce wireless collisions skipping transmission every other round: transmit scheduling.

If the neighbour count is lower than half the number of active slots, both schedules are decreased by one.

Tx scheduling seems a last resort for extremely dense neighbourhoods, and might not occur very frequently. Rx scheduling on the other hand is triggered easily. Consider the same scenario as before: a fully connected network of size N, where we are interested in the probability that one node received the message from another node in the network after a certain number of rounds. Assume that all nodes have the correct neighbour count from the start of the experiment. If $d_i \geq N_s$, a second receive schedule is added, if $d_i \geq 2N_s$, a third. Under the assumptions, every node in the network has the same strategy, although we assume that the round in which the scheduling was started is random. That is, there is no correlation between the receive periods of nodes in a given frame[1].

There are two effects at play: the probability of reception becomes lower (because the transmit and receive slots might not overlap), and the probability of collision becomes lower (because two transmit slots have a lower probability of overlapping). In the analysis without schedules there were only two possible situations each round: success or collision. With the addition of schedules, the absence of collisions does not guarantee success, since the goal node might not be listening.

First analyse only the first round of communication. We now differentiate between the nodes: the node initially transmitting the message under consideration will be the source, the intended destination node the sink, and the other nodes are just neighbours. There are four possible scenarios:

1. The transmission fails due to a collision
2. There is no collision, but at the instant of transmission none of the nodes has an active Rx schedule.
3. There is no collision, but at the instant of transmission the sink node has no active Rx schedule. Some or all of the other nodes do have that.
4. There is no collision, and the sink node has the right Rx schedule

Of these situations, number 1 and 2 lead to an identical scenario in the next round. Number 4 is the situation we are interested in. Number 3 leads to a different set of probabilities in the next round, since each of the nodes that received the message will try to send it in the next round. If any of them succeeds, the transmission is successful.

Let N_{rx} the number of Rx schedules per node (which is assumed to be equal), R_j the Rx schedule for node j in this round, C be the collision event, N_s the number of slots in one schedule, and T_i the transmission slot for node i in this round.

The first possibility is that of a collision. The probability of a collision is reduced by the number of Rx schedules, since the amount of slots that a Tx slot is randomly picked from has increased to $N_s \cdot N_{rx}$. To avoid enumerating all the combinations, we formulate the probability of no collision:

$$\Pr(\neg C) = (1 - \frac{1}{N_s N_{rx}})^{d_i}$$

[1]It would be interesting to verify that this in fact happens in MyriaNed. The scheduling benefits would be ruined if neighbours have a high chance of picking the same sequence of Rx schedules.

Now consider the second case, where there is no collision, but none of the neighbours has an active Rx schedule in the transmission slot. Each neighbour has a chance of $(1 - \frac{1}{N_{rx}})$ of listening in a wrong period:

$$\Pr(\neg(T_0 \in R_j)\forall j \in d_0 \cap \neg C) = (1 - \frac{1}{N_{rx}})^{d_0} \cdot (1 - \frac{1}{N_s N_{rx}})^{d_i}$$

The chance of a direct success (the fourth case) given no collision is the chance that the sink node picks the correct receive schedule, which is the probability that both the sender and receiver have picked the same schedule ($\frac{1}{N_{rx}}$) times the number of schedules.

$$\begin{aligned}\Pr((T_0 \in R_j) \cap \neg C \mid j = \text{sink}) &= N_{rx} \cdot (\frac{1}{N_{rx}})^2 \cdot (1 - \frac{1}{N_s N_{rx}})^{d_i} \\ &= \frac{1}{N_{rx}}(1 - \frac{1}{N_s N_{rx}})^{d_i}\end{aligned}$$

The remaining chances are when one or more of the neighbours receives the message but the sink node does not. With larger neighbourhoods there are many different combinations of successes and failures, and the probabilities become cumbersome. A more feasible approach is looking at the different paths leading to success. We know the success rate for one round. In two rounds, we should also consider the success rate of two-hop paths. Let $S_{i,j,k}$ denote the event of a successful transmission over the path $i \to j \to k$. The probability of a two-hop path via an arbitrary neighbour is:

$$\Pr(S_{0,i,j} \mid j = \text{sink}) = \underbrace{\Pr(\neg C) \cdot \Pr(T_0 \in R_i) \cdot \Pr(T_0 \notin R_j)}_{\text{First round}} \cdot \underbrace{\Pr(\neg C) \cdot \Pr(T_i \in R_j)}_{\text{Second round}}$$

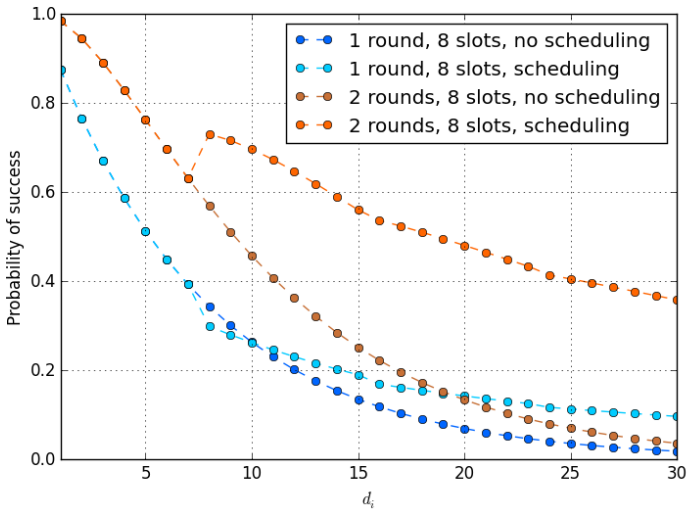
Multiply this by the number of non-sink neighbours $(d_0 - 1)$ to get the two-hop success rate. In two rounds, with scheduling, the total probability of success is thus (1st round success + 1st round failure but 2nd round success + two-hop success).

In Figure B-3a a comparison is made between success probabilities of scheduling and nonscheduling schemes, for one and two rounds, for a maximum of two hops. The probabilities for nonscheduling are steadily decreasing, as expected from Figure 3-6b earlier. The scheduling scheme clearly shows improvement for node densities higher than 7. Indeed, for one round the probability of successful transmission becomes lower, but there is a larger gain in confidence for two rounds. The decision boundary used in MyriaCore is perhaps too high: adding a schedule the first time leads to a sudden jump in success over two rounds. We tried shifting the decision boundary to a lower position (Figure B-3b). This makes the transition of the two-round probabilities smoother, but leads to a significant loss of success in the first round. The first option seems to be the better one.

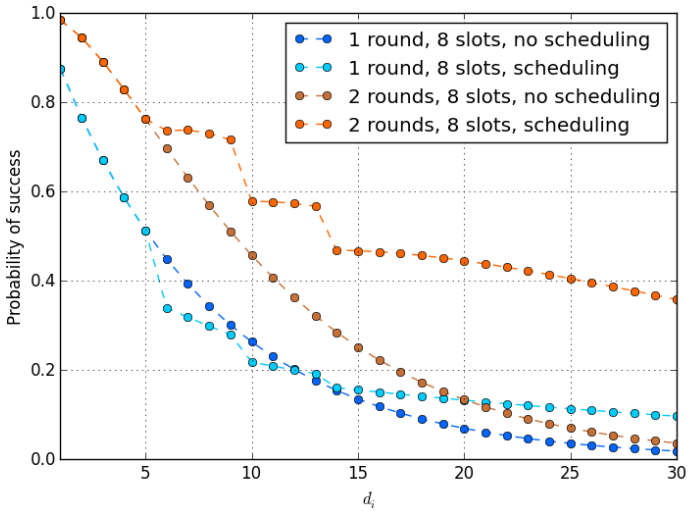
Taking matters to the extreme, we compared the scheduling success with a network that has double the amount of active slots (and thus double the amount of energy consumption!) in Figure B-4. Of course the 16-slot network performs better in most cases. Scheduling seems to be a more efficient approach however, and in extreme cases even works better than just adding more slots.

B-2 Reception Ratios

Now that chances of successful transmission of a single message have been investigated, a second question is the amount of messages that can be expected to be received by a node (possibly the gateway) in a dense neighbourhood. Predicting these number has very practical applications, in judging the packet reception ratios reported by a network, and in estimating the amount of gateways needed for a certain data capacity.



(a) Add Rx schedule for $d > (N_{rx}N_s) - 1$



(b) Add Rx schedule for $d - 1 > \frac{N_{rx} \cdot N_s}{2}$

Figure B-3: The transmission success probabilities for different scheduling regimes, for one or two rounds.

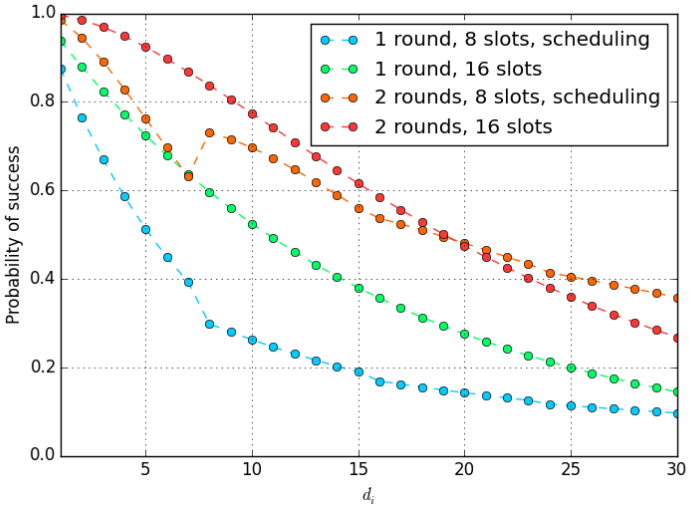


Figure B-4: Comparing success probabilities of a scheduling network with a network with double the amount of slots.

B-2-1 No Scheduling

Consider a fully connected network as before, with N nodes, where for every node the amount of neighbours is $d = N - 1$. There are N_s slots in a round, and each node picks one slot at random in every round to transmit in.

Denote one node in the network as the sink or gateway node. Every slot there is a chance that a message arrives. The message arrival occurs if exactly one neighbour is transmitting in that slot, and the node itself is not sending. In all other cases there is no message arrival.

As calculated earlier, the chance of a successful transmission in a round, by any node in the network is

$$(1 - \frac{1}{N_s})^{N-1}$$

There are N nodes trying to send, so on average the number of successful messages in the network is $N(1-\frac{1}{N_s})^{N-1}$. One of those node is the sink itself however, and can not deliver messages at the sink. So the expected number of received messages by the sink per round is:

$$(N-1)(1 - \frac{1}{N_s})^{N-1} \tag{B-2}$$

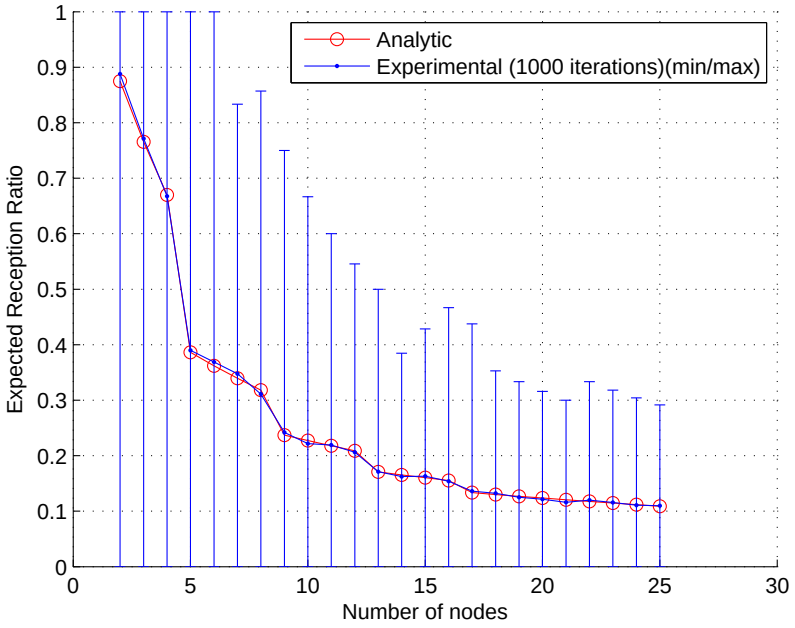
B-2-2 Scheduling

Now we can try to include the influence of scheduling. Assume that all nodes somehow have arrived at all the same number of schedules[2], and their initial schedules are distributed with a uniform likelihood. For example, all nodes have 3 active schedules, and the probability of being in schedule s in a round is $\frac{1}{3}$.

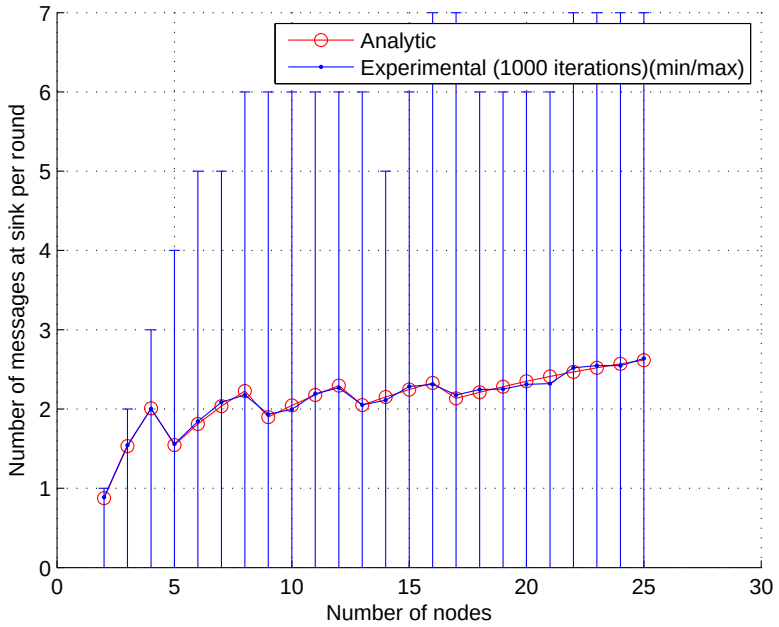
For sending, every node can pick any of the active slots, thus the probability of a message succeeding (the probability of noncollission) is

$$(1 - \frac{1}{N_s N_{rx}})^{N-1}$$

[2]If not, the situation will most likely be worse: nodes are listening when no one is sending, and/or sending when no one is listening



(a)



(b)

Figure B-5: The reception ratios (top) and expected number of messages per round (bottom) for $N_s = 8$, in scheduling mode. The experimental values were retrieved by doing simple simulations of the process.

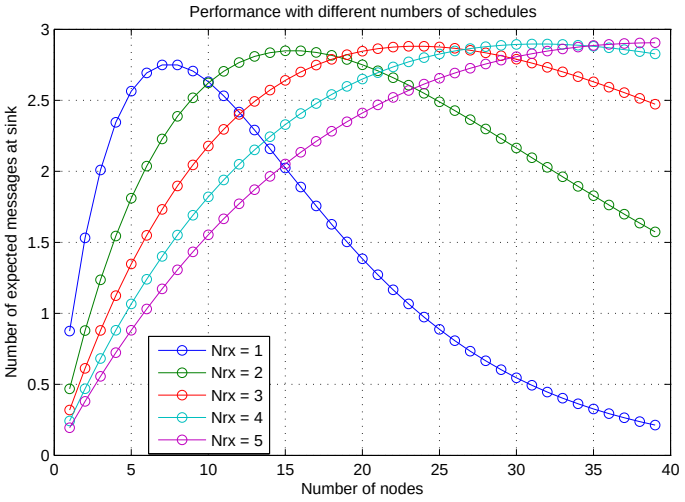


Figure B-6: The influence of the number of schedules on the amount of messages at the sink

This is again done by $(N-1)$ nodes of interest, but there is only a $\frac{1}{N_{rx}}$ chance that the sink is listening. The expected number of messages per round at the sink is now:

$$\frac{N-1}{N_{rx}}(1-\frac{1}{N_s N_{rx}})^{N-1}$$

This formula is identical to equation (B-2) for $N_{rx}=1$, as expected. It is visualised in Figure B-6.

B-3 Concluding

Looking at Figure B-5, we can conclude the following:

- Under optimal circumstances, $N_s=8, d=8$, the reception ratio is indeed around the infamous 36%, but it can be higher when less nodes are around (but that is less efficient). The number of messages received seems to have a maximum somewhere below 3, as confirmed by Figure B-6. In low densities there are not enough neighbours to exceed this, in high densities there are too many collisions.
- In dense neighbourhoods, packet reception ratios are commonly between 10% and 30%.
- The number of messages received at the gateway can be used as a performance indicator for the scheduling decision. In Figure B-6, this measure is plotted for different numbers of schedules, for 8 slots. The crossover points are at 10, 19, 28 and 35 nodes.

B-4 Limits of Throughputs of gMAC and Slotted ALOHA

The probabilities of exactly one message being sent in a slot for gMAC and Slotted ALOHA are:

$$\text{gMAC: } \Pr(X=1)=\frac{m_i+1}{N_s}\cdot(1-\frac{1}{N_s})^{m_i}$$

$$\text{Slotted ALOHA: } \Pr(X=1)=Le^{-L}=\frac{m_i+1}{N_s}\cdot e^{-\frac{m_i+1}{N_s}}$$

Table B-1: The number of neighbours at which point adding an extra schedule will be beneficial.

N_s	2 Schedules	3 Schedules
6	7	13
8	10	18
10	12	23
12	15	28
14	18	33
16	21	38
	$\approx 1.4N_s - 1.4$	$= 2.5N_s - 2$

These functions are very similar, towards the point that in the limits of d to infinity and N_s to infinity, the functions are equal:

$$L = \frac{d+1}{N_s} \rightarrow N_s = \frac{d+1}{L}$$
$$(1 - \frac{1}{N_s})^d = (1 - \frac{L}{d+1})^d$$

The exponential function can be defined by the following limit:

$$e^x = \lim_{n\to\infty}(1 + \frac{x}{n})^n$$
$$\text{thus: } \lim_{d\to\infty}(1 - \frac{L}{d+1})^d = e^{-L}$$

A similar argument can be made for N_s to infinity.

Bibliography

[1] N. Abramson. The ALOHA System - Another Alternative for Computer Communications. In *Proceedings of the Fall Joint Computer Conference*, 1970.

[2] N. Abramson. The ALOHAnet – surfing for wireless data. *IEEE Communications Magazine*, 47(December):21–25, 2009.

[3] P. Anemaet. *Distributed G-MAC*. Msc thesis, TU Delft, 2008.

[4] F. Assegei. *Decentralized frame synchronization of a TDMA-based wireless sensor network*. Msc thesis, Technische Universiteit Eindhoven, 2008.

[5] T. C. Aysal and K. E. Barner. Convergence of Consensus Models With Stochastic Disturbances. *IEEE Transactions on Information Theory*, 56(8):4101–4113, 2010.

[6] E. Boutsma. Alles is verbonden op de CES. *Technisch Weekblad*, Jan. 2015.

[7] R. Carli and S. Zampieri. Network Clock Synchronization Based on the Second-Order Linear Consensus Algorithm. *IEEE Transactions on Automatic Control*, 59 (2):409–422, Feb. 2014. ISSN 0018-9286. doi: 10.1109/TAC.2013.2283742.

[8] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri. Optimal Synchronization for Networks of Noisy Double Integrators. *IEEE Transactions on Automatic Control*, 56 (5):1146–1152, 2011.

[9] R. Carli, E. D'Elia, and S. Zampieri. A PI controller based on asymmetric gossip communications for clocks synchronization in wireless sensors networks. In *IEEE Conference on Decision and Control and European Control Conference*, pages 7512–7517. Ieee, Dec. 2011. ISBN 978-1-61284-801-3. doi: 10.1109/CDC.2011.6161101.

[10] Centraal Bureau voor Statistiek. CBS Statline, 2014.

[11] B. J. Choi, H. Liang, X. Shen, and W. Zhuang. DCS : Distributed Asynchronous Clock Synchronization in Delay Tolerant Networks. *IEEE Transactions on Parallell and Distributed Systems*, 23(3):491–504, 2012.

[12] A. Delawari. *Time Synchronization in Wireless Sensor Networks*. Msc thesis, TU Delft, 2013.

[13] DevLab. Devlab Documentation pages, 2014. URL https://redmine.devlab.nl/.

[14] M. Dobson. *Low-power epidemic communication in wireless ad hoc networks*. PhD thesis, Vrije Universiteit Amsterdam, 2013.

[15] J. C. Eidson. *Measurement, control, and communication using IEEE 1588*. Springer, 2006. ISBN 9781846282508.

[16] J. Elson. *Time synchronization in wireless sensor networks*. PhD thesis, University of California, Los Angeles, 2003.

[17] Y. R. Faizulkhakov. Time synchronization methods for wireless sensor networks: A survey. *Programming and Computer Software*, 33(4):214–226, July 2007. ISSN 0361-7688. doi: 10.1134/S0361768807040044.

[18] R. Fan and N. Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4): 255–266, Jan. 2006. ISSN 0178-2770. doi: 10.1007/s00446-005-0135-6.

[19] P. Frasca, R. Carli, F. Fagnani, and S. Zampieri. Average consensus on networks with quantized communication. *International Journal of Robust and Nonlinear Control*, 19(16):1787–1816, Nov. 2009. ISSN 10498923. doi: 10.1002/rnc.1396.

[20] N. Freris, V. Borkar, and P. Kumar. A model-based approach to clock synchronization. In *Proceedings of the 48th IEEE Conference on Decision and Control, 2009*, pages 5744–5749. Ieee, Dec. 2009. ISBN 978-1-4244-3871-6. doi: 10.1109/CDC.2009.5399516.

[21] N. Freris, S. R. Graham, and P. R. Kumar. Fundamental limits on synchronizing clocks over networks. *IEEE Transactions on Automatic Control*, 56(6):1352–1364, 2011.

[22] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-Sync Protocol for Sensor Networks. In *Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems*, pages 138–149, Los Angeles, CA, 2003.

[23] A. Garulli and A. Giannitrapani. Analysis of consensus protocols with bounded measurement errors. *Systems & Control Letters*, 60(1):44–52, Jan. 2011. ISSN 01676911. doi: 10.1016/j.sysconle.2010.10.005.

[24] D. Gavidia. *Epidemic-style information dissemination in large-scale wireless networks*. PhD thesis, Vrije Universiteit Amsterdam, 2009.

[25] Golledge. CC7V Crystal Oscillator Specifications, 2009. URL http://www.golledge.co.uk/pdf/products/xtl_sm/cc7v.pdf.

[26] J. He, P. Cheng, L. Shi, and J. Chen. Time synchronization in WSNs: A maximum value based consensus approach. In *IEEE Conference on Decision and Control and European Control Conference*, pages 7882–7887. IEEE, Dec. 2011. ISBN 978-1-61284-801-3.

[27] J. He, P. Cheng, L. Shi, and J. Chen. Clock synchronization for random mobile sensor networks. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 2712–2717. Ieee, Dec. 2012. ISBN 978-1-4673-2066-5. doi: 10.1109/CDC.2012.6426053.

[28] J. He, H. Li, J. Chen, and P. Cheng. Study of consensus-based time synchronization in wireless sensor networks. *ISA transactions*, 53(2):347–57, Mar. 2014. ISSN 1879-2022.

[29] F. Heidarian. A Comment on Assegei's use of Kalman Filter for Clock Synchronization of Wireless Sensor Networks – Unpublished. 2010.

[30] F. Heidarian. *Studies on verification of wireless sensor networks and abstraction learning for system inference*. PhD thesis, Radboud Universiteit Nijmegen, 2012.

[31] E. Huet. Google Nest Spoof By German Activists Promises Eerie, Data-Driven Future. *Forbes*, July 2014. URL http://www.forbes.com/sites/ellenhuet/2014/05/07/google-nest-spoof-by-german-activists-promises-eerie-data-driven-future/.

[32] A. Jadbabaie, J. Lin, and A. Stephen Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, June 2003. ISSN 0018-9286. doi: 10.1109/TAC.2003.812781.

[33] H. Karl and A. Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[34] A. Kashyap, T. Başar, and R. Srikant. Quantized consensus. *Automatica*, 43(7): 1192–1203, July 2007. ISSN 00051098. doi: 10.1016/j.automatica.2007.01.002.

[35] B. Krom. Synchronization in a Wireless Sensor Network, Internship Report. Technical report, TU Delft, Chess Wise, 2014.

[36] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. ISSN 00010782. doi: 10.1145/359545.359563.

[37] K. Leentvaar and J. Flint. The Capture Effect in FM Receivers. *IEEE Transactions on Communications*, 24(5):531–539, 1976.

[38] Q. Li and D. Rus. Global Clock Synchronization in Sensor Networks. *IEEE InfoCom*, 2004.

[39] M. Maroti and B. Kusy. FTSP - TinyOS Wiki, 2008. URL http://tinyos.stanford.edu/tinyos-wiki/index.php/FTSP.

[40] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The Flooding Time Synchronization Protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, pages 39–49, Baltimore, MD, USA, 2004. ACM. ISBN 1581138792.

[41] D. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10), 1991.

[42] R. E. Mirollo and S. H. Strogatz. Synchronization of Pulse-Coupled Biological Oscillators. *SIAM Journal of Applied Mathematics*, 50(6):1645–1662, 1990.

[43] L. Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

[44] Nordic Semiconductor. nRF51822 Product Anomaly Notice. Technical report, Nordic Semi, 2014.

[45] Nordic Semiconductor. nRF51822 Product Specification. Technical report, Nordic Semi, 2014.

[46] Nordic Semiconductor. nRF51 Series Reference Manual. Technical report, Nordic Semi, 2014.

[47] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 2006.

[48] R. Olfati-Saber and R. M. Murray. Consensus Problems in Networks of Agents With Switching Topology and Time-Delays. *IEEE Transactions on Automatic Control*, 49 (9):1520–1533, 2004.

[49] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*, 95(1):215–233, Jan. 2007. ISSN 0018-9219.

[50] Peng! Collective. An Open Source Hoax, 2014. URL http://www.google-nest.org/.

[51] S. U. Pillai, T. Suel, and S. Cha. The Perron-Frobenius Theorem. *IEEE Signal Processing Magazine*, 22(March):62–75, 2005.

[52] S. Ping. Delay measurement time synchronization for wireless sensor networks. *IRB-TR-03-013, Intel Research Berkeley Lab*, 2003.

[53] W. Ren and R. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5): 655–661, May 2005. ISSN 0018-9286. doi: 10.1109/TAC.2005.846556.

[54] I.-K. Rhee, J. Lee, J. Kim, E. Serpedin, and Y.-C. Wu. Clock synchronization in wireless sensor networks: an overview. *Sensors*, 9(1):56–85, Jan. 2009. ISSN 1424-8220. doi: 10.3390/s90100056.

[55] L. G. Roberts. ALOHA packet system with and without slots and capture. *ACM SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.

[56] S. Rockman. Google Nest, ARM, Samsung pull out Thread to strangle ZigBee. *The Register*, July 2014. URL http://www.theregister.co.uk/2014/07/15/google_nest_thread_protocol/.

[57] L. Schenato and F. Fiorentin. Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks. *Automatica*, 47(March):1878–1886, 2011.

[58] Seiko Epson Corporation. Fc - 135 / fc - 255 Oscillator Specs. Technical report, Seiko Epson Corporation, 2014.

[59] E. Serpedin and Q. Chaudhari. *Synchronization in Wireless Sensor Networks*. Cambridge University Press, 2009. ISBN 978-0-521-76442-1.

[60] M. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. *IEEE Wireless Communications and Networking*, 2:1266–1273, 2003. doi: 10.1109/WCNC.2003.1200555.

[61] O. Simeone and U. Spagnolini. Distributed Time Synchronization in Wireless Sensor Networks with Coupled Discrete-Time Oscillators. *EURASIP Journal on Wireless Communications and Networking*, 2007(1):057054, 2007. ISSN 1687-1499. doi: 10.1155/2007/57054.

[62] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz. Distributed synchronization in wireless networks. *IEEE Signal Processing Magazine*, 95(September):81–97, 2008.

[63] R. Solis, V. S. Borkar, and P. R. Kumar. A New Distributed Time Synchronization Protocol for Multihop Wireless Networks. *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 2734–2739, 2006. doi: 10.1109/CDC.2006.377675.

[64] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*, 2009. ISBN 9781605583716.

[65] S. H. Strogatz. *Sync*. Hyperion, 2003.

[66] W. Su and I. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, Apr. 2005. ISSN 1063-6692. doi: 10.1109/TNET.2004.842228.

[67] R. Sugihara and R. K. Gupta. Clock Synchronization with Deterministic Accuracy Guarantee. In *7th International Wireless Communications and Mobile Computing Conference*, 2011.

[68] Texas Instruments. TMP112 High-Accuracy, Low-Power, Digital Temperature Sensor. Technical report, TI, 2014. URL http://www.ti.com/product/tmp112.

[69] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications - WSNA '03*, page 11, 2003. doi: 10.1145/941351.941353.

[70] J. R. Vig. Introduction to quartz frequency standards. Technical report, U.S. Army Electronics Technology and Devices Laboratory, 1992.

[71] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. *Computer*, 34(1):44–51, 2001. ISSN 00189162. doi: 10.1109/2.895117.

[72] 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 - SenSys '05*, page 142, New York, New York, USA, 2005. ACM Press. ISBN 159593054X. doi: 10.1145/1098918.1098934.

[73] M. Wohlsen. What Google Really Gets Out of Buying Nest for $3.2 Billion. *Wired*, Jan. 2014. URL http://www.wired.com/2014/01/googles-3-billion-nest-buy-finally-make-internet-things-real-us/.

[74] Y.-c. Wu, Q. Chaudhari, and E. Serpedin. Clock Synchronization of Wireless Sensor Networks. *IEEE Signal Processing Magazine*, 124(January):124–138, 2011.

[75] K. S. Yildirim, R. Carli, and L. Schenato. Proportional-Integral Synchronization In Wireless Sensor Networks – Unpublished. 2014.

[76] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection, 1980. ISSN 0090-6778.

Glossary

List of Acronyms

3mE	Mechanical, Maritime and Materials Engineering
ATS	Average TimeSync
DCSC	Delft Center for Systems and Control
FTSP	Flooding Time Synchronization Protocol
gMAC	gossip MAC
GTSP	Gradient Time Synchronization Protocol
IoT	Internet of Things
MAC	Medium Access Control
MBCS	Model-Based Clock Synchronization
NTP	Network Time Protocol
PTP	Precision Time Protocol
RBS	Reference Broadcast Synchronization
RFA	Reachback Firefly Algorithm
TDMA	Time Division Multiple Access
WSaN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network

List of Symbols

Δf	The drift as a ratio of frequencies (in ppm)
α	The result of a filtered weighted sum of time differences that functions as a drift estimate (but in different units) in MemoryMedian and PISync
β	The general symbol for the weighted sum of errors used for proportional correction in Median, MemoryMedian and PISync

Δ	The maximum degree in the network
$\epsilon_i(k)$	The correction of node i's idle time for round k
τ_i	The value of clock i
$\theta_i(k)$	The set of phase differences with neighbours measured by node i in round k
$\mathcal{E}$	Set of edges in a network
$\mathcal{N}_i$	The set of nodes that node i receives; its neighbours
$\mathcal{V}$	Set of nodes in a network
$A(k)$	Adjacency matrix in round k
a_i	Drift factor of clock i
b_i	Offset of clock i
C	The event of a collisions, a random variable.
d_i	Derived drift value of node $i : T(a_i - 1)$
$f_i(t)$	The actual frequency at time t of the oscillator in node i
f_n	The nominal frequency of oscillators
k_i	The integral gain for a synchronization algorithm
k_p	The proportional gain of a synchronization algorithm
L	The graph Laplacian
m_i	Number of neighbours of node i
N	Number of nodes in a network
N_s	Number of active slots
P	The Perron matrix, $I - k_pL$.
T	The round time, also called frame time of a network
t	Reference time
T_g	The guard times
T_i	The transmission slot for node i, a random variable
$v(k)$	The difference between the median and average at time k
w_{ij}	Weights of links in the adjacency matrix. Indicates a link from j to i if it is nonzero
$x_i(k)$	The phase offset of node i at the start of round k