CS-MNS: Analysis and Implementation

by

Ereth M^{c}Knight-MacNeil

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
August 2010

The undersigned recommend to

the Faculty of Graduate Studies and Research

acceptance of the thesis

CS-MNS: Analysis and Implementation

submitted by

Ereth M^{c}Knight-MacNeil, B.Sc.

in partial fulfilment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Thesis Supervisor, Dr. Thomas Kunz

Chair, Dr. Howard Schwartz
Department of Systems and Computer Engineering

Carleton University

August 2010

Abstract

Wireless sensor networks (WSNs) consist of numerous nodes gathering observations and combining these observations. Often, the timing of these observations is of importance when processing sensor data. Thus, a need for clock synchronization arises in WSNs. The clock sampling mutual network synchronization (CS-MNS) algorithm has been proposed to fulfil this role.

Analytical results are given that show, in the absence of initial offset errors, that the network clocks converge. In the general case, conditions are presented under which the clock rates show convergent behaviour. The CS-MNS algorithm is improved through the addition of a bias term to control initial divergence.

Numerical simulation is used to explore the behaviour of CS-MNS for different network topologies.

The CS-MNS algorithm is implemented under TinyOS and its performance experimentally evaluated. The results show that CS-MNS behaves as pedicted by analysis and simulation, and acheives significantly better clock synchronization performance than the flooding time synchronization protocol (FTSP).

Acknowledgments

I would like to thank my supervisor Dr. Thomas Kunz for his guidance throughout my graduate studies and for his insightful comments and discussions.

I would like to acknowledge the TinyOS community members who have generously shared and donated their work.

I appreciate the support and encouragement I have received from my family over many years. Finally, I am most grateful to Christina Vanderwel for her support, dedication and patience.

Table of Contents

List of Tables

List of Figures

Nomenclature

α_j	Rate of uncorrected clock at node j
α_{min}	Minimum uncorrected clock rate over all nodes in a group
β_{BIAS}	Control bias parameter to control initial divergence
β_j	Offset of uncorrected clock at node j
β_{min}	Minimum clock offset over all nodes in a group
$\epsilon_{i,j}(\tau)$	The error term relating the updated clock rates in the zero-offset and general cases, see Equation 13
$\epsilon_{max}(\tau)$	Absolute maximum of $\epsilon_{i,j}$ over all pairs in a group
ϵ_{quant}	Magnitude of quantization noise introduced by clock granularity
γ	Maximum clock offset difference, corrected for rate differences, over all pairs in a group
λ	Minimum of control gain k and $1 - k$
k	CS-MNS update control gain
N	Number of nodes in group
s_j	CS-MNS correction factor at node j

s_j^+	CS-MNS updated correction factor at node j
$T_j(t)$	Uncorrected clock process at node j at time t
$\widehat{T}_j(t)$	Corrected clock process at node j evaluated at time t
$\widehat{T}_j^+(t)$	Corrected clock process at node j after update evaluated at time t

Acronyms

API application programming interface

CPU central processing unit

CS-MNS clock sampling mutual network synchronization

FIFO first in, first out

FTSP flooding time synchronization protocol

HAA hardware abstraction architecture

HAL hardware adaptation layer

HIL hardware interface layer

HPL hardware presentation layer

I/O input/output

IC integrated circuit

IEEE Institute of Electrical and Electronics Engineers

MEMS microelectromechanical system

MSK minimum shift keying

NTP network time protocol

OS operating system

PC personal computer

ppm parts-per-million

RAM random access memory

RISC reduced instruction set computing

SFD start of frame delimiter

TSF time synchronization function

USB universal serial bus

UTC coordinated universal time

VLSI very-large-scale integration

WSAN wireless sensor and actuator network

WSN wireless sensor network

Chapter 1

Introduction

The possibility of combining microelectromechanical system (MEMS) sensors with very-large-scale integration (VLSI) control, signal processing and communications circuits to form an intelligent wireless sensor node capable of forming wireless networks with like sensor nodes was recognized as early as 1996 in [1]. The 1998 paper by C.J. Pottie entitled, simply, "Wireless Sensor Networks" [2] discusses some of the challenges related to power management and network architecture faced in wireless sensor network (WSN) design. In the considerable subsequent literature on WSNs, various applications have been proposed including intrusion detection, habitat monitoring, and building automation.

Self-organizing WSNs can operate without existing infrastructure. This allows WSNs to be deployed in challenging environments as well as allowing new applications otherwise prohibited by infrastructure costs. However, in order to operate without infrastructure, sensor nodes must rely on batteries for power, which places emphasis on low-power design. Deploying large numbers of nodes requires a low cost for each node. These factors generally dictate WSN nodes with relatively modest computational capabilities, relatively low complexity, and short range radios. Despite the modest capabilities of each node, the goal of WSN design is to design co-operative and distributed protocols where the nodes function together, pooling their resources, to accomplish useful, non-trivial tasks.

With only short range radios available, communication between distant nodes requires forwarding packets through multiple intermediate peer nodes. However, multi-hop peer-to-peer routing combined with duty cycling of radios, changing radio propagation conditions, and node mobility causes relatively long and unpredictable end-to-end transit times for packets. While these long and unpredictable transit times are of minimal importance for collection of long-term sensor data, long and unpredictable transit times make communication of time-sensitive data and timing data difficult in WSNs.

In many WSN applications, sensor readings from multiple nodes need to be combined either to synthesize a snapshot of the conditions at a single point in time or to track the progress of a phenomenon through the sensor field. The challenge of generating a mutually consistent time stamping service across the nodes of a WSN is referred to as time synchronization. In wireless sensor and actuator networks (WSANs), where the passive sensing role of some nodes is augmented with an active role, valid synchronization data is required in real-time to co-ordinate current and near-future actions. Thus, in WSANs post-hoc methods of synchronization are generally insufficient and a more active form of synchronization is required.

1.1 Objectives

Various synchronization algorithms for use in WSNs have been proposed [3]. Among these algorithms is the clock sampling mutual network synchronization (CS-MNS) algorithm proposed in [4–6]. The existing work on CS-MNS was based on simplified analytical analysis and on simulation.

The objective of the work presented here are to:

- Seek a further understanding of the dynamic behaviour of the CS-MNS algorithm.
- Explore implementation issues related to CS-MNS.
- Experimentally evaluate the performance of the CS-MNS algorithm in a testbed WSN environment.
- Experimentally compare the relative performance of the CS-MNS algorithm with existing synchronization algorithm implementations.

For the testbed implementation, TinyOS was chosen as a target operating system and the standard TinyOS flooding time synchronization protocol (FTSP) was chosen as a logical point of comparison.

1.2 Contributions

In keeping with furthering the understanding of the CS-MNS algorithm, we present an analytical analysis of the CS-MNS algorithm that gives a proof of convergence in the case of zero initial offset error. In the general case, we develop bounds outside which CS-MNS continues to show this convergent behaviour. These results lead to an improvement of the CS-MNS algorithm through the addition of a tuning parameter to control initial divergence.

Further investigation of the CS-MNS algorithm was carried out using simplified numerical simulation. These simulations support the analysis by exploring the influence of network topology on the CS-MNS algorithm. Additionally, the simulation results demonstrate the improved behaviour of the CS-MNS algorithm with the addition of the above-mentioned tuning parameter.

The above work resulted in both a refereed conference paper, [7], published in the *Proceedings of the 2009 International Conference on Wireless Communications and*

Mobile Computing: Connecting the World Wirelessly as well as a journal paper, [8], published in *Wireless Communications and Mobile Computing.*

Finally, the CS-MNS algorithm was implemented under TinyOS 2 and tested in a testbed environment. Testing of FTSP was also performed and the relative performance of the two algorithms was compared. In addition to providing an experimental evaluation of CS-MNS and FTSP, the testbed results confirm the analytical and simulation results, showing that all three approaches lead to qualitatively similar behaviour. Explanations for quantitative differences are also discussed.

1.3 Organization of the Thesis

Chapter 2 gives a brief overview of time synchronization and wireless sensor networks and goes on to introduce synchronization algorithms relevant to the thesis.

Results and analysis are presented in Chapters 3, 4, and 5. Chapter 3 provides a theoretical treatment of the CS-MNS algorithm focusing on the convergence properties of the algorithm. In Chapter 4 the results from simulations of CS-MNS presented and discussed. Chapter 5 shows the performance results from testbed-based experiments for both CS-MNS and for FTSP.

Finally, Chapter 6 highlights the conclusions drawn from the work and discusses future work.

Chapter 2

Background

2.1 Wireless Sensor Networks and Their Applications

A wide variety of systems are included under the name of wireless sensor networks (WSNs). However, there are a number of commonalities between WSN systems. First, WSN systems interact with the physical world. This sets WSNs apart from purely computational information systems. Second, WSNs are generally self-organizing and co-operative. These traits are preferred because they allow the WSN to be easily deployed, to avoid or recover from centralized failure, and to operate without pre-existing infrastructure.

Through self-organization and co-operation between nodes, WSNs aim to solve problems using many simple nodes. For example, detailed monitoring of crops, atmospheric conditions and irrigation for agricultural management could be accomplished using a network of battery-powered nodes. The network could be deployed easily and would provide detailed data over a long period. WSNs offer an attractive solution for this application in contrast to the traditional solutions. For example, centralized monitoring of wired senors would incur large up-front installation cost to install the required infrastructure over a wide area. Alternatively, more complex sensors could be deployed to gather the data in a single location such as aerial observation using an infrared camera.

Other applications for WSNs cover a wide range, including: intrusion detection in both civil and military applications [9], wildfire detection and tracking [10], habitat and environmental monitoring [11], and indoor light, temperature, and humidity monitoring for building control [12].

2.2 Wireless Sensor Network Hardware

While wireless sensor network nodes must be both low-cost and low-power, there is a range of hardware platform designs. An example of the more computationally capable hardware is the Imote2 hardware platform. Two examples of less computationally capable hardware are the MicaZ and TelosB designs. The Imote2, TelosB, and MicaZ devices are available commercially from Memsic Corporation and are shown in Figure 1. All three of these devices use the CC2420 intelligent radio integrated circuit produced by Texas Instruments. This radio provides IEEE 802.15.4 radio communications in the 2.4 GHz ISM band at a bit rate of 250 kbps [13, 14].

The CC2420 intelligent radio includes hardware AES-128 encryption and presents a serially-connected packet-level interface to the micro-processor. Additionally, the radio integrated circuit (IC) supplies a separate hardware timing strobe that is triggered during each packet pre-amble whenever a packet is transmitted or received. This allows the host micro-controller to perform a hardware-level time stamp of outgoing and incoming packets. The transmitting and receiving radios operate at a symbol rate of 62.5 kHz, where each symbol consists of 32 chips for a chip rate of 2 MChips/s. The offset between the I and Q phases in the minimum shift keying (MSK) modulation scheme used hints at synchronization between the transmitter and receiver sampling

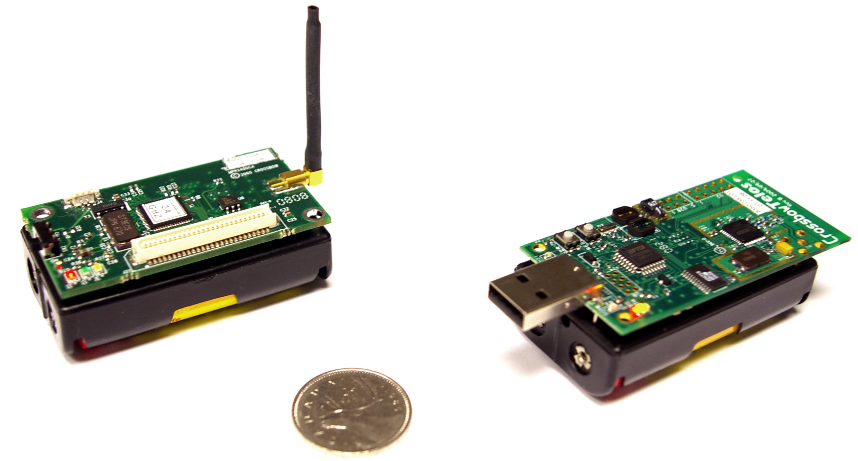


Figure 1: MicaZ (left) and TelosB (right) sensor node hardware.

clocks on the order of 0.5 μs. A delay of 2 μs between the receiver timing edge and the transmitter timing edge is specified for the CC2420 [15].

Both the MicaZ and TelosB device platforms use simple reduced instruction set computing (RISC) micro-controller designs targeted at low-power, low-cost embedded applications. The older MicaZ design uses the 8-bit ATmega128L processor from Atmel running at 7.3728 MHz. On-board the Atmel processor has 4 kB RAM for data memory and 128 kB flash for program storage. The TelosB design uses the 16-bit MSP430 processor from Texas Instruments running at 4 MHz. Again, on chip memory is used with 10 kB data random access memory (RAM) and 48 kB program flash. Neither design includes any floating point hardware although both processors include multiplication hardware [16, 17].

The TelosB and MicaZ platforms each have a number of different oscillators. However, each platform limits the oscillators that can be used for software-level timing. On the TelosB platform the 4 MHz central processing unit (CPU) clock oscillator is a low-accuracy digitally controller oscillator implemented as a ring oscillator. This oscillator is trimmed by software at boot time using the 32.768 kHz crystal oscillator as a reference. Thus, although the 4 MHz oscillator has higher resolution than the 32.768 kHz clock, the frequency accuracy cannot exceed that of the 32.768 kHz clock since it is effectively derived from this source. The MicaZ does implement the 7.3728 MHz CPU clock as a crystal oscillator. However, on both systems the CPU oscillators are stopped in low-power sleep states. This leaves only the 32.768 kHz oscillator capable of providing a continuous time reference.

2.3 Wireless Sensor Network Software

In an effort to make the TinyOS code bases modular and reusable, the code base is written in a dialect of C called NesC. NesC adds object-oriented style modularization on top of plain C. However, in contrast with other traditional object-oriented languages, NesC is designed to resolve all object references at build time and creates a statically linked executable. Thus, NesC results in object-oriented style code but without the run time indirection overheads associated with object references [18, 19].

The modular nature of NesC is used to implement a three-layer hardware abstraction model which is explained in Appendix A.

As many of the target platforms for TinyOS lack memory management hardware, the operating system (OS) also lacks protected memory concepts. In TinyOS there is no distinction between kernel space and user space. Instead, TinyOS is implemented as a collection of interdependent NesC modules. By creating a dependency on a user written module, the TinyOS user causes his or her code to be included along with the OS code in the TinyOS build process.

2.4 Clocks

Clocks measure the passing of time, both electronic and mechanical clocks do so by counting the integrating the periods of an oscillator. In the case of a mechanical clock, the oscillator often takes the form of a pendulum or a mass and a spring, while the integration function is performed by the clock hands. In electronic clocks, the oscillator is often a tuned crystal circuit, although any oscillator circuit may be used, while the integration function is performed by a digital counter. Crystal oscillator circuits are particularly attractive for use in clocks because the oscillation period depends primarily on the natural frequency of the crystal.

However, the natural frequency of a crystal is controlled by controlling the physical size and shape of the crystal. Thus, the crystal period is subject to error related to the physical manufacturing tolerances for the crystal. Unfortunately, crystal are not perfectly stable over very long time periods or across temperature variation.

Tolerance in the frequency of crystals introduces rate error to a clock. In lay terms, if two clock exhibit rate error, one clock will gain time relative to the second clock. Another type of error is clock offset. Clock offset error is the familiar everyday error exhibited by a clock that is, ‘five minutes fast.’ A clock which is either ahead or behind of another clock by a constant time increment.

Even with rate and offset errors a clock remains linear, but in practise clocks exhibit non-linear behaviours. However, these non-linearities can often be ignored for clocks with millisecond and microsecond resolutions.

2.5 Time Synchronization

The basic concept of time synchronization as a means that ensures a common notion of time between multiple parties is one that is familiar from everyday experience. In everyday life clocks are often used simply to agree on a common time, for example the time to convene a meeting or the timing of a train departure. While everyday clocks nominally reflect the timing of the earth’s rotation, this is mostly irrelevant for scheduling the start of a meeting. For example, daylight saving time introduces an arbitrary adjustment twice annually, but when those involved all make the same adjustment, the utility of their clocks as a common notion of time is preserved.

The concept of a common timebase shared within a group but unconnected to any outside reference is termed relative synchronization or internal synchronization. In contrast, maintaining a common timebase synchronized with an outside reference, such as coordinated universal time (UTC), is termed absolute synchronization or external synchronization. By necessity, absolute synchronization implies relative synchronization.

Different problems require different types of synchronization. One common use for time synchronization is the ordering of events, which requires only relative synchronization. In fact, Lamport gives restrictions required for correctness in this situation [20] that are less strict than even relative time synchronization. In other applications the choice between absolute and relative synchronization is less clear. For example, time-of-flight based acoustic target tracking [21] requires time synchronization where the epoch is unimportant but where the time units must be related to those of the physical world. However, in practice relative synchronization with arbitrary time units may be substituted for absolute time units as long as the localization errors introduced by this substitution are sufficiently small for the particular application.

In order to synchronize individual clocks, some clocks must be allowed to influence others in the group. Systems of coupled clocks naturally raise questions of stability and group convergence. There is a body of theoretical work based on the Kuramoto model of phase coupled oscillators, one review of this work is given in [22]. However, even in the Kuramoto model the relation between coupling and the onset of synchronization is complex and a number of open problems remain.

In wired networks, numerous synchronization protocols have been developed and deployed. Of note is the network time protocol (NTP) as described in [23]. Protocols such as the NTP sidestep the complexity of coupled oscillator dynamics by enforcing a hierarchy among the clocks. In NTP, the hierarchy is organized into levels or strata. Each node uses timing data only from nodes in lower numbered or equal numbered strata. This approach enforces a one-directional flow of synchronization data and prevents feedback loops from forming.

2.6 Time Synchronization in Wireless Sensor Networks

Time synchronization is an important foundation of networked systems. In particular, many WSNs rely upon distributed clocks to allow correct analysis of collected sensor data. However, the ad hoc and dynamic nature of WSN topologies prevents the straight-forward application of traditional centralized, hierarchical clock synchronization strategies. Limited energy availability and cost sensitivity pressures further limit the application of traditional algorithms [24].

Even if algorithms designed for wired networks will work in WSNs, these algorithms may fail to take advantage of the nature of WSNs. For example, algorithms designed for point-to-point networks fail to capitalize on the broadcast nature of the wireless medium. Additionally, traditional algorithm designs may be poorly optimized for the nature of WSNs. For example, mobility among WSN nodes can cause frequent changes in the network topology, which might cause some algorithms to expend excessive overheads as configuration tasks are repeated again and again to adapt to each change in network topology. In response to this limited applicability and in keeping with the properties generally found in WSNs, various self-organizing centralized and decentralized algorithms for WSN synchronization have been proposed [3]. Below, three synchronization protocols of relevance to this thesis are reviewed.

2.7 IEEE 802.11 Time Synchronization Function

The Institute of Electrical and Electronics Engineers (IEEE) 802.11 time synchronization function (TSF) is widely implemented and used when an IEEE 802.11 network is operating in an ad hoc mode. The IEEE 802.11 TSF is extremely simple and similar to the method put forth by Lamport in [20]. However, as explained in [25] and [26], the IEEE 802.11 TSF suffers from poor scalability as all of the requirements given by Lamport in [20] are not met in the case of IEEE 802.11 ad hoc networks.

The IEEE 802.11 TSF uses regularly timed periodic beacons. The start of each beacon period consists of a contention window. During the contention window each node schedules a beacon transmission after a random back off period. Once a node receives a beacon, the node cancels the pending beacon transmission. Ignoring collisions and bias caused by rate error among the back-off timers, this mechanism nominally allows one beacon transmission per beacon period transmitted from a randomly selected node.

Each beacon contains the time value at the transmitting node. Once a node receives a beacon, the node compares the time value in the beacon with the time value local to the node. If the beacon time value is greater the node resets the local time to be equal to the beacon time value. Otherwise, the receiving node ignores the beacon. The IEEE 802.11 TSF makes no attempt to synchronize the individual node clock rates and instead relies on periodic re-adjustment in order to prevent the accumulation of large errors.

In [27], Zhou and Lai given an “industry expectation” that “the maximum clock offset be under 25μs for an” ad hoc group independent of network size. However, they indicate that for a large ad hoc group the “offset between stations can be over 4000 μs” for groups using the 802.11 TSF.

2.8 Flooding Time Synchronization Protocol

The TinyOS distribution contains an implementation of the flooding time synchronization protocol (FTSP) described in [28, 29]. The FTSP protocol operates by selecting a single root node which serves as the group time reference. This root node periodically broadcasts its local clock. Receiving nodes gather a series of broadcasts and use a linear least squares solution to estimate the rate and offset errors of their local clocks. Once a node has made this estimate, the node begins to act as a repeater, sending out periodic synchronization message. Each node continues to update its error estimates by dropping the oldest data points from the regression.

The master node is selected dynamically as the node with the lowest network address. Every message contains the current root node address. Upon receipt, each node will update the local root node variable if the incoming root node address is lower than the local variable. Each node compares the local root node address with the node's own address to detect if the node should act as the root node. However, to protect against scenarios in which the root node becomes isolated from all or part of the network, each node will reset the local root node variable if no messages with this root address are received for a number of message periods. This has the effect of allowing a new root node to be selected when the root becomes disconnected from the network.

FTSP broadcast messages contain a sequence number which is incremented by the root for each broadcast. Each node tracks the highest sequence number received and ignores any packets with a sequence number equal to or lower than the highest sequence number seen. In well-connected networks a node may receive many messages during each beacon period as each neighbouring node floods the synchronization data. However, the sequence number filtering causes each node to ignore all but the first message received each period.

In [29], the authors give some performance on Mica2 hardware using a 7.37 MHz clock as the clock source. The authors give a maximum error of less than 14 μs in a network consisting of 60 nodes with a maximum distance of 6 hops from the root node. This maximum error value increased to 67 μs when the root node was switched off and a new root was selected, transforming the network into a 59 node network with a maximum distance of 11 hops to the root node.

However, in [30] the authors present a sensor network model with uniformly distributed jitter in message timing measurements. The authors show that under this model FTSP exhibits synchronization error that grows exponentially with network diameter. Furthermore, in [30] the authors show both simulation and testbed results consistent with their theoretical treatment.

2.9 Clock Sampling Mutual Network Synchronization

The clock sampling mutual network synchronization (CS-MNS) algorithm is presented in [4–6]. The CS-MNS algorithm has previously been shown through simulation to perform well and has been shown analytical to exhibit stability in specific conditions [6].

The CS-MNS algorithm is fully decentralized in that all nodes execute the same algorithm at all times. Furthermore, the algorithm does not require knowledge of, and makes no assumptions about, the network topology. These properties allow CS-MNS to be applied easily in randomly deployed networks and in dynamic networks. Furthermore, the algorithm requires no additional overhead or energy to run adaptation procedures in response to changing radio propagation conditions nor in response to nodes leaving the network through battery depletion or otherwise.

As originally presented, the CS-MNS algorithm uses periodic beacons with a beacon contention mechanism equivalent to the IEEE 802.11 TSF. The simple beacon format of CS-MNS is compatible with IEEE 802.11 TSF beacon format. The CS-MNS algorithm can be implemented in software with standard IEEE 802.11 radio and clock hardware [6]. This makes CS-MNS applicable for use in networks of currently available hardware and in cost-sensitive consumer devices that must use commodity radio hardware.

The CS-MNS algorithm uses a single multiplicative correction factor to correct the local clock. Unlike FTSP, which holds a series of samples as well as other housekeeping variables, the correction factor is the only state held by the CS-MNS algorithm. Upon receiving any beacon the CS-MNS algorithm updates the local correction factor. Only if the local time value and the time value in the received beacon are identical will the correction factor remain unchanged. Thus, in contrast to the IEEE 802.11 TSF and to FTSP, the data in every received beacon is used to update the CS-MNS clock correction.

Two analytical results for CS-MNS are presented in [6]. Both results are applicable only in the absence of communication errors and only in the absence of initial offset error. The first result, shows that under these conditions any number of slaves exhibit locally asymptotic stability toward a point where the slave node clocks become equal to the master node clock. However, this master-slave case assumes that the master node makes no clock adjustment, limiting the applicability of the result in deployed systems.

The second analytical result in [6] gives sufficient conditions, again with the absence of communication and initial offset errors, for the CS-MNS algorithm to exhibit local asymptotic stability toward a point where the corrected time processes at the two nodes are equal. However, the paper leaves the “case with any number number of nodes and arbitrary topology” as “an open problem” [6]. Further investigation in [6] is performed through simulation and maximum errors of 19 μs and 32 μs for single hop groups of 100 and 500 nodes, respectively, are reported.

2.10 CS-MNS Clock Correction

Each node maintains a hardware clock which is allowed to run freely at its natural rate. At any point in time, t, we represent this uncorrected hardware clock at node j by $T_j(t)$. However, in order to synchronize the nodes the CS-MNS algorithm must create a new, correct clock. CS-MNS uses a simple multiplicative transformation to generate this corrected clock from the uncorrected clock. As given in [4–6, 31] the corrected clock, $\widehat{T}_j$, is related to the uncorrected clock, T_j by

$$\widehat{T}_j(t) = s_j T_j(t) \tag{1}$$

and the CS-MNS algorithm controls the correction factor s_j.

The CS-MNS algorithm modifies the correction factor by comparing the local corrected time with the corrected time, $\widehat{T}_i$, sampled from some other node i. Based closely on the CS-MNS update law as given in [4–6, 31] the updated correction factor, s_j^+ is calculated as

$$s_j^+ = s_j + k\frac{\widehat{T}_i(\tau) - \widehat{T}_j(\tau)}{T_j(\tau)} \tag{2}$$

where k represents a control gain and τ is the time at which the clock sample is taken. Implicitly, it is assumed that the clocks at node j and node i are sampled at the same instant in time τ. While this cannot be achieved exactly in practise, the hardware support for radio message time stamping described in Sections A.3 and 2.2 allows for the two node clocks to be sampled well within a single clock period.

By substituting the updated correction factor into Equation 1, the new time estimate, $\widehat{T}_j^+$, is given as

$$\widehat{T}_j^+(t) = \left(1 + k\frac{\widehat{T}_i(\tau) - \widehat{T}_j(\tau)}{s_j T_j(\tau)}\right) s_j T_j(t) \tag{3}$$

which can be simplified by using Equation 1 again to yield

$$\widehat{T}_j^+(t) = (1-k)\,\widehat{T}_j(t) + k\frac{\widehat{T}_i(\tau)}{\widehat{T}_j(\tau)}\widehat{T}_j(t) \tag{4}$$

where the distinction between t, the independent time variable, and τ, the clock sample time, must be made carefully.

As discussed in [6], the update law in Equation 2 makes proportional corrections. The magnitude of the change made to the correction factor s_j is proportional to the magnitude of the error between the local and remote clock samples. However, the denominator in Equation 2 dictates that the magnitude of the adjustment is inversely proportional to the elapsed time. Thus, the CS-MNS algorithm makes large corrections when errors are large but makes progressively smaller, finer adjustments as synchronization progresses.

The CS-MNS algorithm takes as input a number of clock values all sampled at the same instant. The corrected local, $\widehat{T}_j$, and remote, $\widehat{T}_i$, clocks and the uncorrected local clock, T_j, are used as input. The stored state of the correction factor, s_j is also used. The CS-MNS algorithm then outputs an updated correction factor, s_j^+, which can be used to generate an updated corrected clock, $\widehat{T}_j^+$. These new values are adopted as the input for the subsequent iterations of the algorithm.

2.11 Summary

Wireless sensor networks present numerous design challenges and open a number of areas of research. These area include routing algorithms, extremely low power system design and optimization, and clock synchronization. The CS-MNS algorithm provides a promising clock synchronization approach consistent with the overall intent of WSN design. The CS-MNS algorithm is democratic, distributed, and has low memory and computational complexity. Further insight into the behaviour of the CS-MNS algorithm is desirable. Additionally, experimental measurements of CS-MNS performance and comparison with other algorithm performance is desirable.

Chapter 3

Analytical Results

3.1 Introduction

The analytical analysis of the clock sampling mutual network synchronization (CS-MNS) algorithm begins by outlining a simplified model of a real-world clock. This model clock model is then used throughout the remainder of the analysis. The CS-MNS clock correction scheme and the CS-MNS adaptation law are explained and an expression for the updated clock in terms of previous clocks is derived.

The theoretical convergence properties of the CS-MNS update law are examined both in the special case of zero offset error and in the general case. Unconditional asymptotic convergence results are obtained in the case of zero offset error. Limits are derived under which the general case exhibits similar convergence properties to the special case.

A small improvement to the CS-MNS update law is presented that discourages divergence during the beginning of the synchronization period. Finally, the effect of quantization noise on the CS-MNS algorithm is discussed briefly.

3.2 Analytical Clock Model

For the purpose of analysis we adopt a simplified clock model exhibiting only offset and rate errors. Such a clock is termed an affine clock in [32] because the model presents the clock at each node as an affine transformation of a theoretical, prefect reference clock. Thus, the time process at node j is modelled by

$$T_j(t) = \alpha_j t + \beta_j \tag{5}$$

where t represents the reference time process. Implicitly, the clock rates and offsets, α_j and β_j, remain constant in time.

The justification for adopting the affine clock model stems from the assumption that the other forms of clock error are small in relation to the clock offset and rate errors. Thus motivated, the affine clock model can be interpreted as the Maclaurin series expansion of the true clock truncated after the first order term. The approximation that offset and rate error dominate overall clock error is part of the motivation behind the design of CS-MNS.

The applicability of the affine clock model to the hardware clocks on TelosB and MicaZ sensor nodes is explored experimentally in Section 5.4.

Under this model, the corrected time process given in Equation 1 becomes,

$$\widehat{T}_j(t) = s_j T_j(t) = s_j \left(\alpha_j t + \beta_j\right) \tag{6}$$

at node j.

3.3 Convergence in the Absence of Offset Errors

The convergence properties of the CS-MNS update law are a natural area of investigation. Some insight into the behaviour of CS-MNS can be motivated by considering the special case of synchronizing clocks without offset errors. This special case corresponds to the condition that all $\beta_1 = \cdots = \beta_n = 0$ in Equation 5.

In this case the corrected time process of Equation 6 becomes simply

$$\widehat{T}_j(t) = s_j T_j(t) = s_j \alpha_j t \tag{7}$$

and substituting this into the expression for the updated corrected time of Equation 4 yields

$$\widehat{T}_j^+(t) = (1-k)\, s_j \alpha_j t + k \frac{s_k \alpha_k \tau}{s_j \alpha_j \tau} s_j \alpha_j t \tag{8}$$

which simplifies to

$$\widehat{T}_j^+(t) = (1-k)\, \widehat{T}_j(t) + k \widehat{T}_i(t) \tag{9}$$

for this case. The updated corrected clock rate is also of interest and is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\widehat{T}_j^+ = s_j^+ \alpha_j = (1-k)\, s_j \alpha_j + k s_k \alpha_k = (1-k)\frac{\mathrm{d}}{\mathrm{d}t}\widehat{T}_j + k\frac{\mathrm{d}}{\mathrm{d}t}\widehat{T}_i \tag{10}$$

in the absence of offset errors.

If the control gain k is kept in the range $k \in (0,1)$, then Equation 9 expresses the updated time estimate at node j as a strict convex combination of the previous time estimates at notes j and k. Furthermore, Equation 10 shows that the updated clock rate at node j is also a convex combination of the previous clock rates at nodes j and k. This is exactly the requirement given as Assumption 1, part 3 by Moreau in [33].

The intuitive motivation for the argument formalized in [33] stems from the observation that an update law with this form cannot increase the worst-case level of disagreement within the group. From Equation 10, the updated rate must satisfy

$$\min_l (s_l \alpha_l) \leq s_j^+ \alpha_j \leq \max_l (s_l \alpha_l) \tag{11}$$

which simply states that any updated rate must lie between the minimum and maximum rates in the group. Thus, we can immediately conclude that the minimum and maximum rates for the group with updated rates are either unchanged or have moved toward each other.

However, while illustrative, the intuitive explanation does not offer insight into what conditions might allow updates to continue with a constant degree of asynchrony. For this analysis we turn to Moreau's analysis in [33]. The updated clock rate given in Equation 10 satisfies the requirements of Assumption 1 in [33]. Thus, in [33] the proof of Theorem 1 proves the above intuitive statement that the level of disagreement cannot be increasing.

Theorem 2 of [33] states that a necessary and sufficient condition for the system to be "uniformly globally attractive with respect to the collection of equilibrium solutions" [1], where the equilibrium solutions are constant, is for there to be a $T \geq 0$ such that there is a node that is connected to all other nodes across all time periods $[t_0, t_0 + T]$. This condition can be interpreted as saying that there must be a finite length of time such that any period of this length contains at least one node which can spread information to all other nodes.

Thus, applying Theorem 2 of [33] results in the conclusion that the CS-MNS update applied to a collection of node clocks without offset error will, with sufficient communication, continuously drive the node clock rates towards a common rate close to the initial rates that does not change with time. The result that the equilibrium point, toward which the group approaches, is not changing in time is not intuitively

[1] The precise definition of uniform global attractivity in this case is defined by Moreau in Appendix I of [33].

obvious.

Finally, we conclude, that in the absence of offset error convergence of clock rates implies convergence of clocks. However, in the case without offset errors the clocks converge toward a rate bounded by the fastest and slowest initial clock rates in the group. Thus, the clocks converge toward relative synchronization where the final group clock is independent of any external absolute standard.

3.4 Convergence with Offset Errors

While the above theoretical results for the special case without offset errors are heartening, it is unlikely that a group of clocks would have zero offset error in practise. Thus, we examine the effect of introducing offset error into the above analysis.

We proceed by adding an error term to the updated clock rate given by Equation 10 in the zero-offset case so that it is equal to the rate from the general case given in Equation 4,

$$\frac{\mathrm{d}\,\widehat{T}_j^+(t)}{\mathrm{d}t} = (1-k)\,s_j\alpha_j + ks_i\alpha_i + ks_i\alpha_i\epsilon_{i,j}(\tau) \tag{12}$$

where the error term,

$$\epsilon_{i,j}(\tau) = \frac{\beta_j - \frac{\alpha_j}{\alpha_i}\beta_i}{\alpha_j\tau + \beta_j} \tag{13}$$

arises from the non-zero values of β_i and β_j.

If $s_j\left(\tau\right)\alpha_j < s_i\left(\tau\right)\alpha_i$ and

$$-\,k\big(s_i\left(\tau\right)\alpha_i - s_j\left(\tau\right)\alpha_j\big) < ks_i\alpha_i\epsilon_{i,j}\left(\tau\right) < (1-k)\,\big(s_i\left(\tau\right)\alpha_i - s_j\left(\tau\right)\alpha_j\big) \tag{14}$$

or $s_j\left(\tau\right)\alpha_j > s_i\left(\tau\right)\alpha_i$ and

$$-\,(1-k)\,\big(s_i\left(\tau\right)\alpha_i - s_j\left(\tau\right)\alpha_j\big) < ks_i\alpha_i\epsilon_{i,j}\left(\tau\right) < k\big(s_i\left(\tau\right)\alpha_i - s_j\left(\tau\right)\alpha_j\big) \tag{15}$$

then the updated clock rate in Equation 12 is strictly between $s_i\alpha_i$ and $s_j\alpha_j$. In other words, despite the error term, the updated clock rate will lie in the interior of the range between the previous corrected clock rates. Thus, under the above conditions the essential property that led to convergent behaviour in the case without offset error is preserved.

By defining

$$\lambda = \min\big(1-k, k\big) \tag{16}$$

then the above conditions will always be met if

$$|\epsilon_{i,j}(\tau)| < \frac{\lambda}{k}\left|\frac{s_j\alpha_j - s_i\alpha_i}{s_i\alpha_i}\right| \tag{17}$$

regardless of whether node i or j has the slower clock. Further, by choosing

$$\begin{aligned} \alpha_{min} &= \min_i \alpha_i \\ \beta_{min} &= \min_i \beta_i \\ \gamma &= \max_{i,j} \left|\beta_j - \frac{\alpha_j}{\alpha_i}\beta_i\right| \end{aligned}$$

then $|\epsilon_{i,j}(\tau)|$ can be bounded over all i, j as

$$|\epsilon_{i,j}(\tau)| \leq \epsilon_{max}(\tau) = \frac{\gamma}{\alpha_{min}\tau + \beta_{min}} \tag{18}$$

which becomes smaller with increasing time. Indeed, $\lim_{\tau\to\infty} \epsilon_{max}(\tau) = 0$.

Finally, using $\epsilon_{max}(\tau)$ and Equation 17 we arrive at the condition

$$\epsilon_{max}(\tau) < \frac{\lambda}{k}\min_{i,j}\left|\frac{s_j\alpha_j - s_i\alpha_i}{s_i\alpha_i}\right| \tag{19}$$

under which all possible CS-MNS rate updates exhibit the required properties for convergent behaviour. Thus, regardless of network topology, any period in which all relative clock rate errors are greater than a threshold will be periods in which the clock rates show convergent behaviour. In addition, Equation 18 shows that this threshold grows smaller as time passes.

It is worth noting that while Equation 19 can fail to be satisfied if only a few nodes have very similar clock rates, Equation 17 is stronger. In cases where Equation 19 is not satisfied among all nodes, the updates between nodes with largely differing clock rates will still satisfy Equation 17 for each update. Thus, in practise many nodes continue convergent behaviour even when Equation 19 cannot be used to guarantee convergent behaviour of the system.

The viewpoint presented so far has served well for developing the above condition but in terms of the behaviour of a wireless sensor network (WSN) the condition in Equation 17 requires that any two nodes with relative skew greater than some threshold will adjust their clocks to reduce this skew. Thus, it is possible for clusters of clocks with very little relative clock skew to make updates based on one-another's clocks under conditions that do not satisfy Equation 17. Thus, multiple clusters could conceivably resist adjusting their rates towards the other clusters.

However, the definition of $\epsilon_{max}(\tau)$ in Equation 18 shows that this threshold grows smaller as time passes and tends towards zero in the limit. This is consistent with the intuitive argument that as time passes more and more of the clock error is a result of clock skew and the relative contribution of the offset errors becomes smaller.

We note that the extreme nodes that define the maximum and minimum rates within a group are the most likely to satisfy the condition in Equation 17.

From the above analysis we can gain some insight into an optimal values for the control gain k. Considering Equations 14 and 15 we observe that the limits are symmetric in the case when $k = 0.5$. Thus it is not surprising that $k = 0.5$ is the value that maximizes the $\frac{\lambda}{k}$ term in Equation 17. Thus, $k = 0.5$ serves to maximize the range of $\epsilon_{i,j}$, and thus the magnitude of offset errors, that can be tolerated while the CS-MNS control law will make updates placing the updated clock rates between the existing clock rates. However, identifying optimal values for k remains an open problem.

3.5 Addition of a Bias Term to Control Initial Convergence

When synchronization begins, the value of τ in Equation 13 is small, which leads to a large value for $\epsilon_{i,j}(\tau)$. Because of this large value, updates are less likely to satisfy the convergence condition Equation 19 initially. This causes divergent behaviour until $\epsilon_{i,j}(\tau)$ shrinks sufficiently.

However, an initial divergent period can serve to further desynchronize the node clocks, which puts the algorithm at a disadvantage once convergent behaviour emerges. In order to prevent this initial divergent behaviour, a bias factor can be introduced into the update law in Equation 2 to arrive at

$$s_j^+ = s_j + k\frac{\widehat{T}_i(\tau) - \widehat{T}_j(\tau)}{T_j(\tau) + \beta_{BIAS}} \tag{20}$$

when β_{BIAS} is a non-negative constant parameter expressed in units of time.

With this updated control law the expressions for the error term and error term bound become

$$\epsilon_{i,j}(\tau) = \frac{\beta_j - \frac{\alpha_j}{\alpha_i}\beta_i}{\alpha_j\tau + \beta_j + \beta_{BIAS}} \tag{21}$$

and

$$\epsilon_{i,j}(\tau) \leq \epsilon_{max}(\tau) = \frac{\gamma}{\alpha_{min}\tau + \beta_{min} + \beta_{BIAS}} \tag{22}$$

which are consistent with the previous definitions when $\beta_{BIAS} = 0$.

The system designer can choose an appropriate value for β_{BIAS} by imposing the requirement that initial updates should exhibit convergent behaviour. If all possible initial updates satisfy Equation 17, where $\epsilon_{i,j}(\tau)$ includes a β_{BIAS} term, then the initial updates will show convergent behaviour. If the designer somehow knew the s, α, and β parameters for each update the designer could solve Equation 17 for values of β_{BIAS} that would encourage convergence.

Since the designer does not know which nodes will update, the designer can approximate by removing the node dependence by using $\epsilon_{max}(\tau)$ on the left hand side and by introducing expected values for the differences in offset and rate in Equation 17 to arrive at

$$\frac{\mathbb{E}\left[\left|\beta_j - \frac{\alpha_j}{\alpha_i}\beta_i\right|\right]}{\alpha_{min}\tau + \beta_{min} + \beta_{BIAS}} < \frac{\lambda}{k}\frac{\mathbb{E}\left[\left|s_j\alpha_j - s_i\alpha_i\right|\right]}{\max_i\left(s_i\alpha_i\right)} \tag{23}$$

which leaves only values that the designer would typically be able to estimate. The expected values and extrema of clocks can be estimated from the expected distributions of the clocks at start-up. For example, the expected and maximum rate error of crystal clocks are easily evaluated from the crystal specifications. By substituting these estimates, along with with a value for τ representing the time of the initial update of interest, the system designer can solve for a value of β_{BIAS} that is expected to result in initial convergence.

A less rigorous estimate, than the estimate in Equation 23, that appears useful in practise is to eliminate the expected value by replacing them with the maximum offset and rate errors and to assume that $\beta_{min} = 0$ while $\frac{\alpha_j}{\alpha_i} = s_i\alpha_i = 1$, leading to

$$\beta_{BIAS} > \frac{k}{\lambda}\frac{\max\left|\beta_j - \beta_i\right|}{\max\left|s_j\alpha_j - s_i\alpha_i\right|} - \tau \tag{24}$$

which allows a simple estimation of an appropriate β_{BIAS}. When the rate errors are measured in parts per million and the initial offset errors are small, the above approximation provides the system designer a rough guide to selection of a β_{BIAS} value.

Replacing the expected value of the clock rate errors with a maximum on the right hand side of Equation 23 is more logical than it initially appears because this chooses the minimum β_{BIAS} value so that the two outlier clocks will have convergent updates. While other clocks may not converge initially, the outlier clocks are expected to begin squeezing the group towards synchrony.

3.6 Effect of Quantization Noise on the CS-MNS Update Law

When implemented in as a digital system, there is necessarily some quantization error in the timestamps used to drive the synchronization algorithm. If timestamps are sampled to within $\pm\epsilon_{quant}$ and the quantization errors are independent, the CS-MNS update from Equation 20 can be re-written with quantization errors as

$$\widetilde{s_j^+} = s_j + k\frac{\widehat{T_i}(\tau) - \widehat{T_j}(\tau) \pm 2\epsilon_{quant}}{T_j(\tau) + \beta_{BIAS} \pm \epsilon_{quant}} \tag{25}$$

which can be expressed as

$$\widetilde{s_j^+} = s_j^+ \pm k\frac{2\epsilon_{quant}}{T_j(\tau) + \beta_{BIAS} \pm \epsilon_{quant}} \mp k\frac{\widehat{T_i}(\tau) - \widehat{T_j}(\tau)}{\left(T_j(\tau) + \beta_{BIAS}\right)^2 \pm \left(T_j(\tau) + \beta_{BIAS}\right)\epsilon_{quant}} \tag{26}$$

in terms of the updated correction factor without errors, s_j^+. Ignoring the higher order terms, an estimation of the quantization errors added to the value of s_j by an update is $\pm 2k\frac{\epsilon_{quant}}{T_j(\tau)+\beta_{BIAS}-\epsilon_{quant}}$.

Another benefit of the β_{BIAS} term is seen here as it serves to reduce the quantization errors added to the running value of s_j during the initial period when $T_j(\tau)$

is small. Again, the system designer can check the magnitude of the β_{BIAS} term by ensuring that the above equation for noise added to s_j while $T_j(\tau)$ is small is on the same order of magnitude as the expected initial clock rate error.

3.7 Summary

We have introduced the affine clock model used throughout the work that motivates the CS-MNS correction scheme. We have introduced the CS-MNS update law and shown that in the case of zero offset errors this update law guarantees convergence of the clock rates. Further to this we have outlined conditions under which this convergent behaviour remains in the presence of offset errors.

The CS-MNS update law was refined by adding a bias term that serves to discourage initial divergence of the clock rates. We have given simplified expressions that allow a network designer to calculate a suitable bias term depending on data plausibly available to the designer at design time. Additionally, we have briefly discussed the quantization noise introduced by clock granularity under CS-MNS.

Chapter 4

Simulation Results

4.1 Introduction

Investigation into the dynamics of clock sampling mutual network synchronization (CS-MNS) is continued through simulation. In paticular, the effect of the network topology is of interest. The analytical results, while applicable because they make only weak assumptions regarding the shape of the connectivity graph, offer little information about how network topology might effect the group dynamics. Additionally, through simulation we seek to confirm the applicability of the analytical results.

In order to investigate the dynamics of the CS-MNS update law easily, a few simplifying assumptions are adopted. The goal of this initial investigation is not to match the real-world performance of the algorithm, but instead to numerically test the update law itself.

The beacon transmission and processing is assumed to be instantaneous. The simulation is conducted under the assumption that all of the nodes broadcast beacons as a homogeneous Poisson process with rate λ_{node}. This implies that, considering a group of N nodes, the composite stream of beacons is also a Poisson process with rate $\lambda = N\lambda_{node}$. Taken together with the assumption that the beacon transmission is instantaneous, the Poisson process assumption eliminates the possibility of beacon collision.

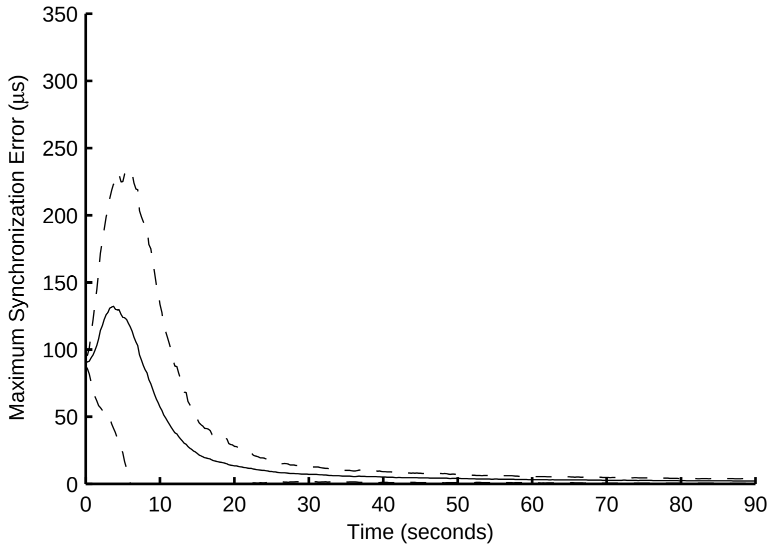
Furthermore, the memorylessness property of the Poisson processes implies that each beacon in the composite beacon stream has equal probability of originating from any node. Thus, the composite beacon stream can be simulated numerically by generating exponential inter-arrival delays with mean $\mu = \frac{1}{\lambda}$ and uniformly assigning a source node from the group.

These simplifying assumptions allow large groups of nodes to be simulated quickly. However, since the analytical arguments make the assumption of continuous state variables and thus ignore the clock granularity, it is of interest to simulate the clock granularity. In this work, the simulation was implemented in MATLAB. MATLAB provides both an integrated development environment and integrated plotting functions.

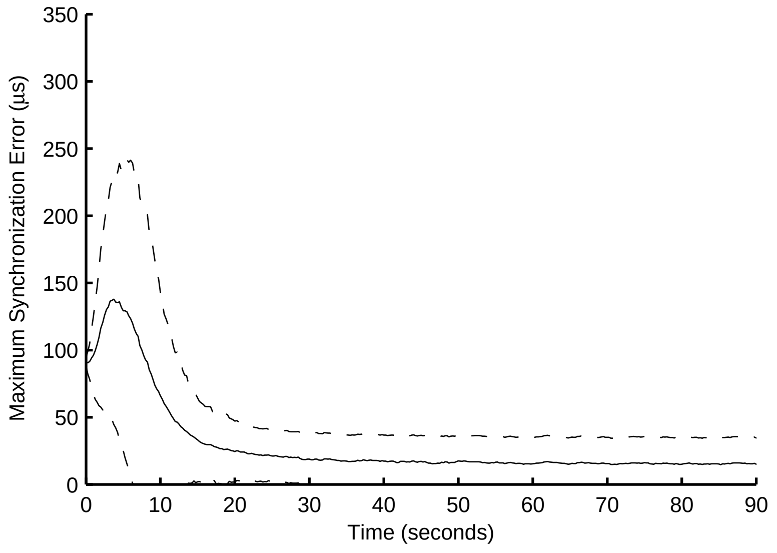
4.2 Simple Dynamics Simulation

Under the above assumptions, simulations were run using floating point time values at each node as well as with discrete 32.768 kHz beacon values. In both cases the total number of nodes was 30 in a single-hop topology. The node clocks were initially uniformly distributed within 92 μs which corresponds to three ticks of a 32.768 kHz clock. The node clock rates were uniformly distributed in the range of ± 50 parts per million. The total beacon rate was one beacon per second, the control gain was $k = 0.5$, and a bias value of $\beta_{BIAS} = 20\,000$ ticks was used. This bias value is the value suggested by Equation 24 for the above initial offset error and range or clock rates.

Each case was simulated for 1 000 repetitions and the mean as well as the 95% confidence interval for the maximum error among the nodes are shown in Figure 2. Clock quantization was applied to the beacon only, with the errors being calculated



(a) Continuous Clocks



(b) 32.768 kHz Clocks

Figure 2: Simulated synchronization error for 30 nodes in single-hop configuration using (a) continuous node clocks and (b) 32.768 kHz quantized node clocks. The solid lines represent the mean values while the dashed lines represent the 95% confidence intervals.

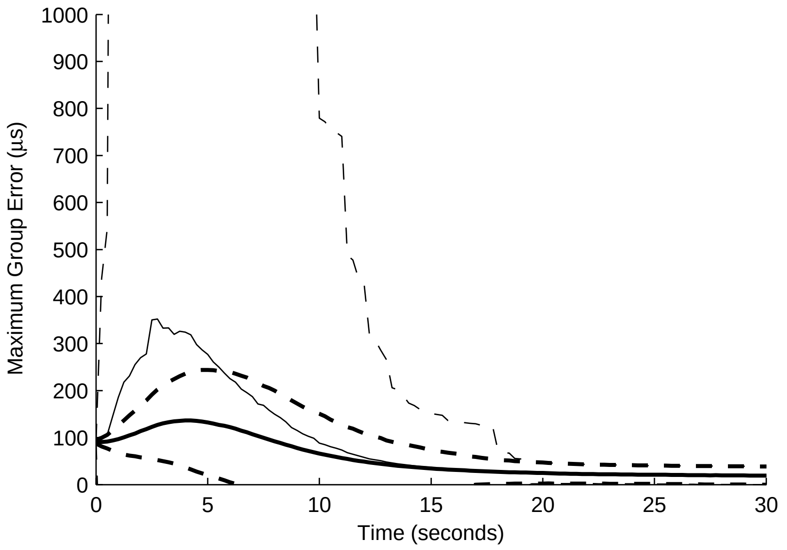


Figure 3: Simulation results showing results without bias term (thin traces) and with a bias value of $\beta_{BIAS} = 20\,000$ ticks (heavy traces). The solid lines represent the mean values while the dashed lines represent the 95% confidence intervals.

exactly. From the graphs the initial behaviour can be seen to be unaffected by the quantization of the node clocks. However, in the granular clock case the mean error settles at about 21 μs after the first 25 seconds. As the value of 21 μs corresponds to just less than the time quanta of a 32.768 kHz clock, persistence of disagreement at this level is within expectation. In the continuous node clock case the simulation shows the level of disagreement continuing to decrease and by the end of the simulation at three minutes the error is approximately 2 μs.

It is worth pointing out that after the first half minute of run time the expected number of beacons from each node is one for a total of 30 beacons transmitted. This is a small communication cost, equal to the communications cost required to compute the group mean for a fixed value.

Figure 3 shows the effect of adding the bias term to the CS-MNS update law. The simulation with bias is identical to the simulation in Figure 2(b). The simulation without the bias term maintains all other parameters. The results shown in Figure 3

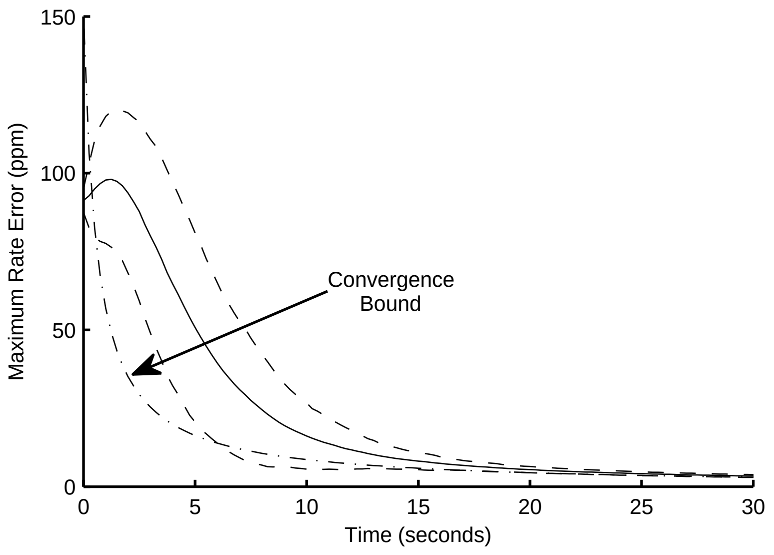


Figure 4: Simulated rate error for a 30 node single-hop topology with convergence bound. The convergence bound from Equation 21 is shown (dash-dot line). The solid line represents the mean value while the dashed lines represent the 95% confidence interval.

are based on 10 000 repetitions both with bias value and without. Again, the group is a 30 node single-hop network. The node clocks were initially uniformly distributed within 92 μs which corresponds to three ticks of a 32.768 kHz clock. The node clock rates were uniformly distributed in the range of ± 50 parts per million. The beacon rate at each node was $^1\!/_{30}$ beacons per second.

The simulation was run using the same underlying random numbers for initial conditions and beacon timing both with and without bias term. Thus, the results are directly comparable. Figure 3 shows that the bias term reduces the initial divergence of the CS-MNS algorithm. While the peak of the mean across the trials for the synchronization error in the case without bias is slightly more than twice that for the case with bias, the confidence interval for the case without bias is very wide indicating a large variance among the trials during the initial convergence period.

Shown in Figure 4 is the maximum rate error among 30 nodes in a single-hop configuration. The bias value is $\beta_{BIAS} = 20\,000$ ticks while the control gain $k =$

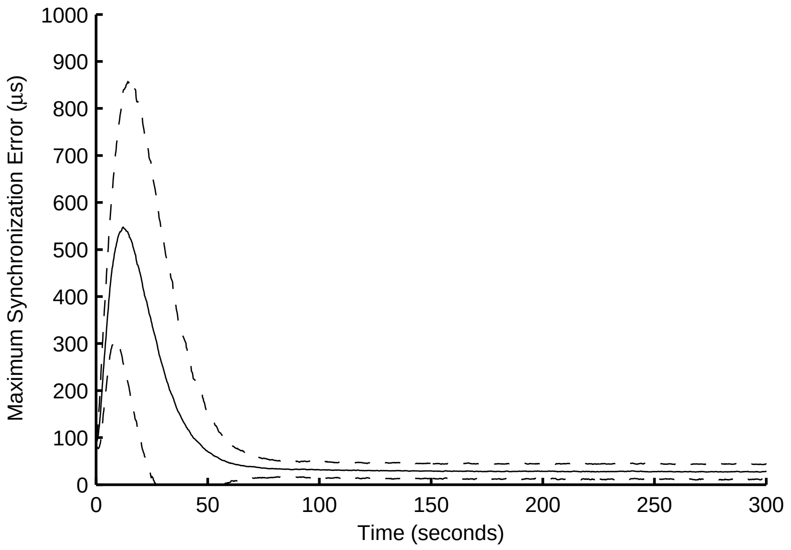


Figure 5: Simulated synchronization error for a 30 node network with each beacon received by 10 randomly chosen nodes. The solid line represents the mean value while the dashed lines represent the 95% confidence interval.

0.5. The convergence bound is shown calculated from Equation 21 based on the initial 92 μs offset errors. The results are based on 2500 repetitions. Initially the rates diverge, but soon after crossing the bound line the rates begin to converge. Beyond 6 seconds the 95% confidence interval contains the convergence bound. As the simulation progresses, the rate error lower confidence bound converges to the convergence bound.

Figure 5 shows the results of a network with randomly changing topology. The results are based on 1 000 repetitions of the simulation. Each beacon is received by a random subset of 10 nodes instead of all 30 nodes, as would be the case in a single-hop network. The random subset of receiving nodes changes with each beacon. This scenario can be interpreted as either a single-hop network with a high packet loss ratio or as a network of very mobile nodes.

This scenario clearly demonstrates that the CS-MNS algorithm does not rely on receiving multiple consecutive messages from the same node and can bring about

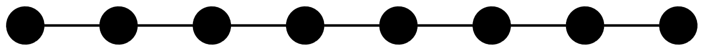


Figure 6: Network topology for the multi-hop line configuration.

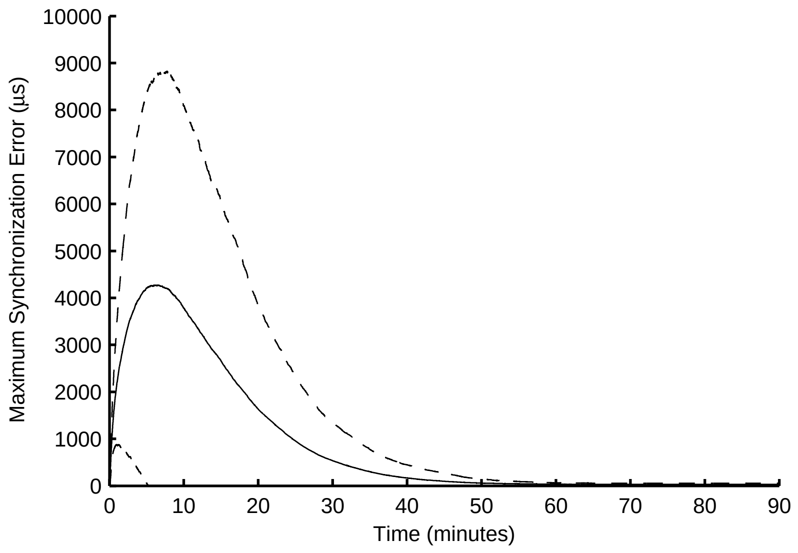


Figure 7: Simulation synchronization error for the multi-hop line configuration. Note that the independent axis is shown in minutes. The solid line represents the mean value while the dashed lines represent the 95% confidence interval.

cooperative behaviour even among nodes with only sporadic communication. Any system requiring long setup phases would be extremely difficult to implement in such a scenario.

Figure 7 shows the results of an 8 node multi-hop network arranged in a single line. The network topology is shown in Figure 6 and communications is possible only between adjacent nodes. The beacon rate at each node is $^1/_{30}$ beacons per second. The results are based on 1 000 repetitions of the simulation. After 70 minutes the synchronization error confidence region settles centred around 30 μs, which is approximately the node clock granularity simulated. The level of synchronization attained compares favourably with the single-hop case. However, in the multi-hop line configuration a much longer period is required to attain this synchronization and the peak synchronization error is much larger.

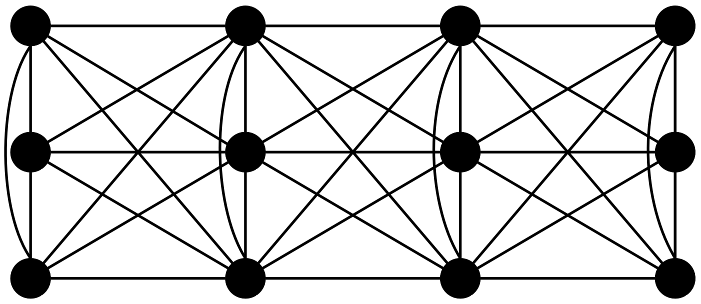


Figure 8: Network topology for the multi-hop group configuration.

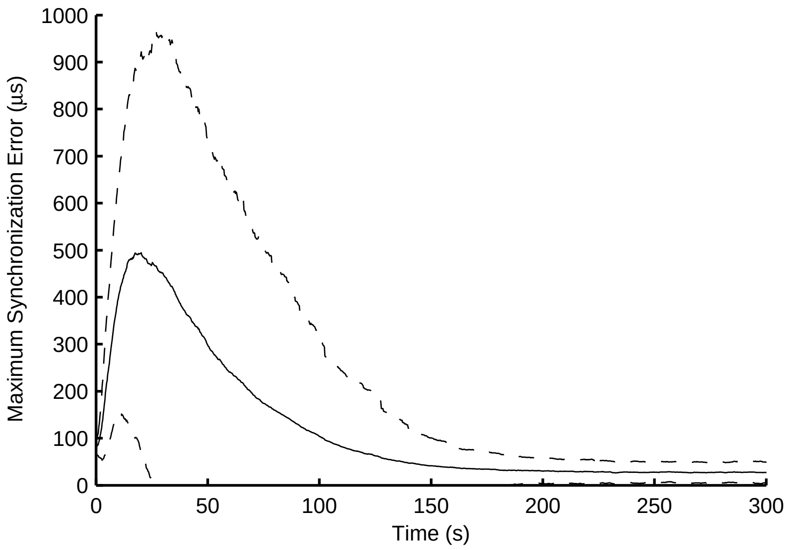


Figure 9: Simulated synchronization error for the multi-hop group configuration. The solid line represents the mean value while the dashed lines represent the 95% confidence interval.

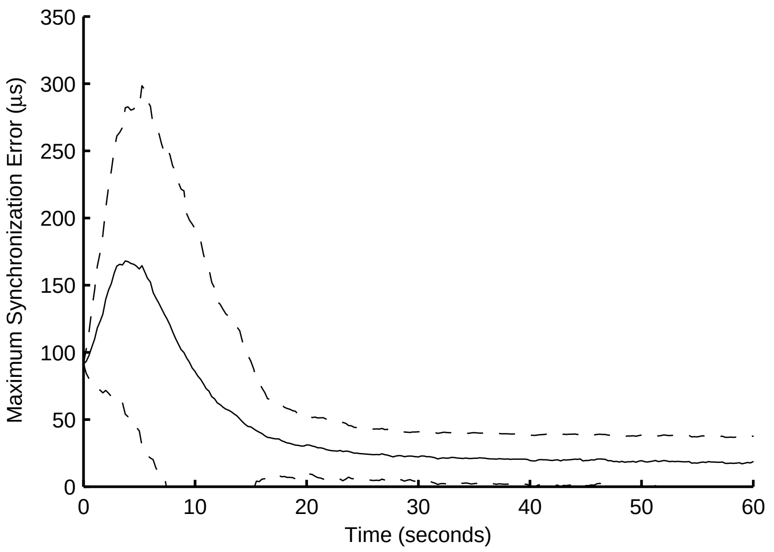


Figure 10: Simulated synchronization error for a single-hop group of 500 nodes. The solid line represents the mean value while the dashed lines represent the 95% confidence interval.

Another multi hop scenario was tested in which the nodes were arranged in subgroups of three nodes, where all nodes within a subgroup can communicate. Four of these subgroups were arranged in a line such that any node in each subgroup could communicate with the nodes in adjacent subgroups. This network topology is shown in Figure 8. The simulation results are shown in Figure 9 and are based on 1 000 repetitions.

Comparing with Figure 7, the overshoot in the group scenario is very small in the multi hop group scenario as compared to the multi hop line scenario. While there are more nodes in the group scenario, the network diameter is only 3 hops, whereas the network diameter in the line scenario is 7 hops.

Figure 10 shows the perfomance of CS-MNS in a large single hop group of 500 nodes. The beacon rate at each node was slowed to $^1/_{500}$ beacons per second to yield a total beacon rate of one beacon per second. Again the initial clock alignment was uniform within 92 μs and initial clock rates were distributed uniformly within ±50 parts per million. A bias value of 20 000 ticks was used. The results are based on 500 runs. The mean synchronization error converges to approximately 30 μs which is on par with the quantization error expected from the clock.

These results are slightly better than those that would be expected from a simplistic master-slave scheme using the same one beacon per second communications cost. Also of note is that with an average beacon rate of one beacon per second only approximately 6% of the nodes can be expected to have transmitted within the first 30 seconds yet the CS-MNS algorithm has significantly improved the group synchronization among all nodes by this time.

The performance of CS-MNS in a large group is significant because, as discussed in Section 2.7, Institute of Electrical and Electronics Engineers (IEEE) 802.11 time synchronization function (TSF) is known to perform poorly in large groups. Additionally, the distributed design of CS-MNS renders the algorithm complexity insenstive to the number of nodes participating, making CS-MNS a possible candidate for synchronization in large networks.

Shown in Figure 11 is the simulated synchronization error for a 14 node single-hop group where half of the nodes have an additional 14% error added to the $\pm$50 ppm error. The simulation was run with a value of $\beta_{BIAS} = 20\,000$ tick in keeping with previous simulations although Equation 24 would dictate a different value if the system designer knew of the large additional rate error. This simulation is motivated by the hardware characterization results in Section 5.4 where a large, somewhat unexpected error was discovered between two hardware platforms. The synchronization error after one minute is only 45 μs, which is larger than the case without the additional large rate error but is small compared to the 8 second error that would be expected without synchronization. However, the peak synchronization error is almost 260 ms, which is three orders of magnitude larger than the peak error seen in the case without the additional rate error.

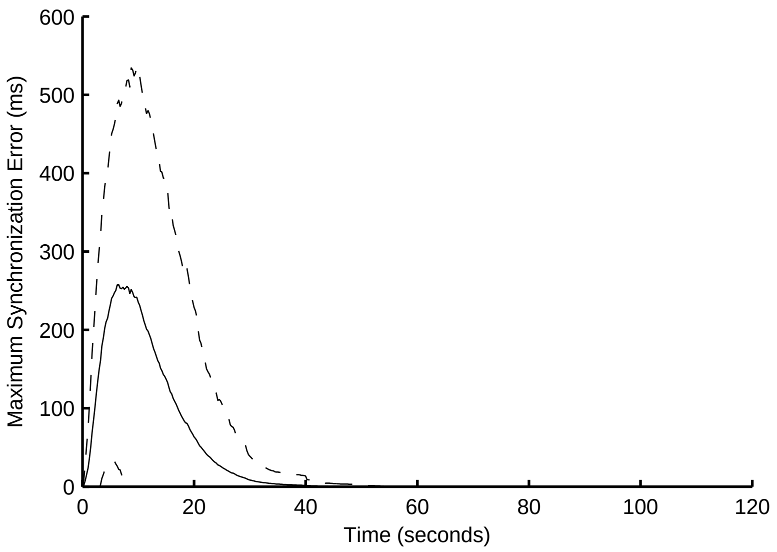


Figure 11: Simulated synchronization error for a 14 node single-hop topology where half of the nodes have a 14% clock rate error. Note that errors are shown in milliseconds. The solid line represent the mean value while the dashed lines represent the 95% confidence interval.

Topology	Synchronization Error		
	Maximum (μs)	Final (μs)	Time to Final (s)
Single-Hop (Cont. Clock)	134	2[a]	16[b]
Single-Hop	137	21	25
Multi-Hop Group	498	28	250
Random Subset Comm.	558	28	150
Multi-Hop Line	4 267	30	4 200
Single-Hop w/ Clock Skew	259 000	45	60

[a] Value after 90 s. Convergence continuous until end of simulation.
[b] Time to achieve 21 μs error. Convergence continuous until end of simulation.

Table 1: Summary of CS-MNS simulation results under different topologies.

4.3 Summary

A summary of the performance of CS-MNS across different topologies, as indicated by simulation, is given in Table 1. While the total number of nodes is not the same in each case, some conclusions can still be drawn. The different scenarios have a large effect on both the peak synchronization error and the time required to achieve a steady state error, but the final synchronization achieved shows less sensitivity across the scenarios. In no scenario did CS-MNS fail to synchronize the group.

Additionally, simulation shows that, in the single-hop case, the bias term proposed can control the magnitude of the initial divergence as well as the sensitivity to initial clock parameters. Across all of the scenarios simulated, the general shape of the CS-MNS response was consistent. The general shape of the synchronization error consisted of an initial divergence followed by a period of convergence. This convergence then slowed, and in the case of a quantized clock, approached a steady state value near the clock quantization interval. In the case of a continuous clock the convergence continued but at a very slow rate. In all scenarios the simulation results showed a single peak synchronization error.

In all of the scenarios using a quantized clock the results showed a final steady state error of similar magnitude. Thus, the initial performance and the rate of convergence of the CS-MNS algorithm is controlled by the initial clock parameters, the bias parameter, and the network topology and is largely insensitive to the clock granularity. Meanwhile, the final synchronization attained is largely controlled by the magnitude of the clock quanta.

Chapter 5

Testbed Results

5.1 Introduction

In order to verify the simulation results and to compare the clock sampling mutual network synchronization (CS-MNS) algorithm against the existing TinyOS implementation of the flooding time synchronization protocol (FTSP) algorithm, the CS-MNS algorithm was implemented under TinyOS 2.1 and experiments were conducted. The initial stage consisted of validating the experimental setup and characterizing the node hardware by collecting clock data from nodes with free-running clocks. Once this was completed, the CS-MNS and FTSP algorithms were tested in a number of different single-hop and multi-hop network configurations. Additional tests of the CS-MNS algorithm were run to confirm the effect of adding the bias term as predicted by analysis and simulation.

5.2 Implementation

The CS-MNS module uses the hardware radio packet time stamping capabilities of the underlying hardware to record departure and arrival times of synchronization packets. The low-level implementation details are discussed in Appendix A.

The CS-MNS clock adjustments were implemented using fixed-point arithmetic, using eight bits for the whole part and 56 bits to the right of the radix point for a total width of 64 bits. This allows for corrections smaller than one part in one quadrillion (smaller than 10^{-15}) without resorting to floating point arithmetic. This is significant since our target hardware lacks hardware floating point support and software floating point libraries demand significant memory footprint and results in calculation inefficiencies.

CS-MNS does not require assumptions about the order of beacon transmissions, which allows CS-MNS to be implemented with various underlying beacon scheduling disciplines. However, in keeping with the simple, democratic nature of the CS-MNS concept, a probabilistic, unco-ordinated beacon schedule was chosen. In this schedule, beacon transmission at each node is a Poisson process. This is the same transmission schedule used in simultion.

5.3 Test Methodology

Tests are conducted by designating a node as the test initiation node and a node as base station node. The test initiation node used was a TelosB node because the TelosB design provides a user push button connected to a micro-controller input. The base station node is connected to a desktop personal computer (PC) using either the universal serial bus (USB) port directly, if the base station is a TelosB, or, in the case of a MicaZ base station, by mounting on the MIB520 programming adaptor. In either case, the node appears to the PC operating system as a USB-connected serial port. The code in the base station node uses the TinyOS `printf` functionality to communicate with the PC.

The TinyOS `printf` functionality consists of a library that implements the C `printf` text formatting function on the node and sends the packetized function output

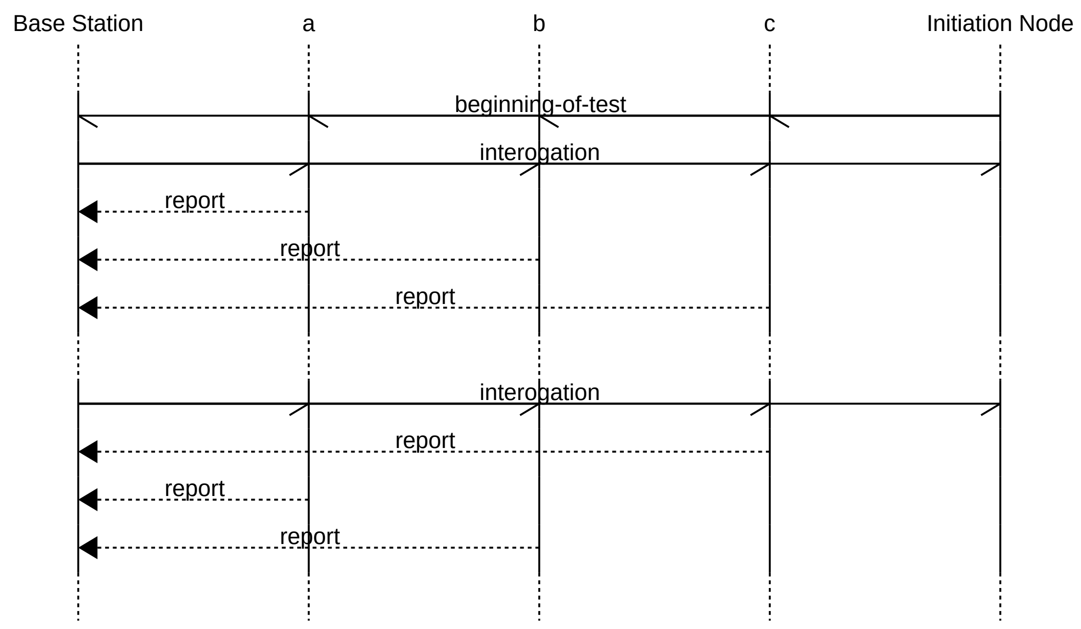


Figure 12: Testbed message sequence showing three nodes ('a', 'b', and 'c') under test.

over USB-connected serial link to the PC. A companion client program on the PC depacketizes this data and outputs the data to a system input/output (I/O) stream. The `printf` output from the node can then be inspected on the terminal or piped to a file.

The base station node program outputs a test log using the `printf` functionality to output test measurements in comma-separated form. This creates a simple flat file data log when the output of the `printf` client is piped to a file.

Once the test nodes are physically arranged and powered, the operator begins the test by pressing the push button on the test initiation node. In response, the initiation node broadcasts a beginning-of-test message. One the base station node, the low-level time stamping application programming interface (API) is used to record the receive time of this beginning-of-test broadcast and this time stamp is sent to the PC for recording.

Upon receiving the beginning-of-test message, nodes begin to run the tested synchronization protocols. In the case of CS-MNS, this beginning-of-test message also serves to fulfil the initial synchronization requirement for starting the algorithm. If required in order to test algorithm performance with poor initial synchronization, individual nodes add a uniformly distributed random value to the local clock value.

Once the test has begun, the base station node periodically sends an interrogation broadcast messages. Each of these messages contains a sequential message identifier. The base station node sends the transmission time stamp of these heartbeat messages to the PC for recording. The message sequence during a test is shown in Figure 12.

Upon receiving an interrogation message, a receiving node under test notes the incoming time stamp as measured by the local node clock. After waiting for a random back-off period, the receiving node replies with a measurement message addressed to the base station node containing its local time stamp. The base station node then reports these timestamps to the PC for recording.

In a single-hop environment, the simple process explained above is sufficient to allow all nodes to reliably report local node timestamps. The large randomized back-off period before attempts to submit a time stamp report minimizes channel contention and any channel capture issues.

The relative uniformity of the reporting method can be seen in Figure 13. Ideally, each of the 3 787 beacons would result in a successful report from each of the nodes; representing over 60 000 packets. However, some packets are lost due to signal interference or other effects and an optimal result would be for these losses to be uncorrelated with node number. In Figure 13 the data is ordered to show the natural grouping of MicaZ and TelosB hardware. It is pertinent to note that the base station node was of the MicaZ design. Also apparent in the data is that TelosB node `a95d` submitted the fewest reports with only 2 483 successful reports. Table 2 shows the mean and standard deviations for various groupings of the nodes. These statistics

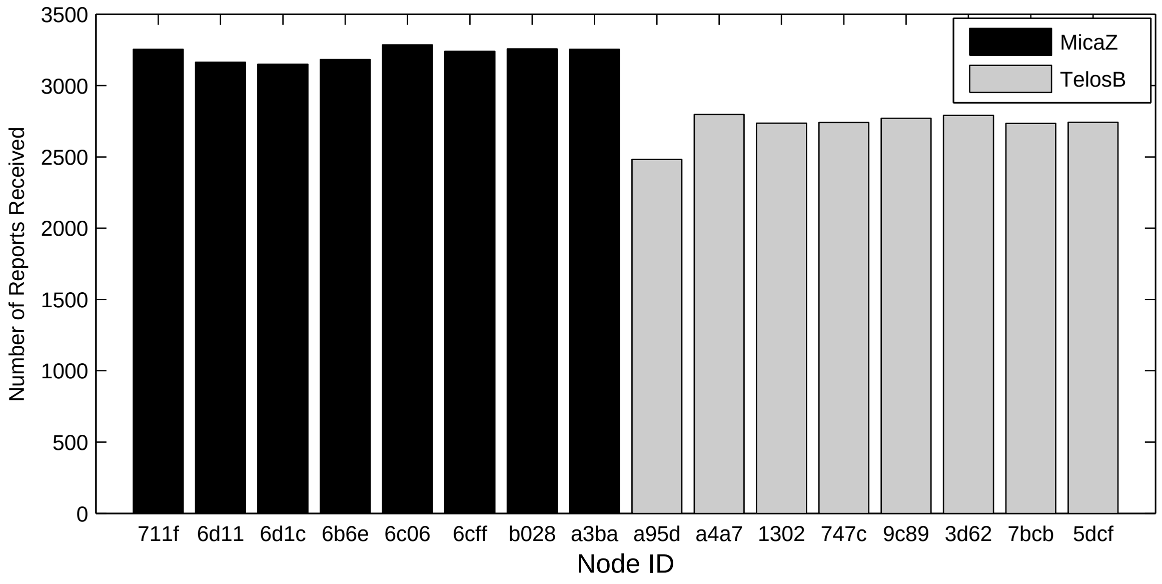


Figure 13: Number of reports received by nodes in a 16 node single-hop network. Statistics for this data are shown in Table 2.

Reports per Node	Mean ($\bar{x}$)	Standard Deviation (σ)
All	2 974	268.8
MicaZ	3 224	50.3
MicaZ without node `6d1c`	3 234	43.8
TelosB	2 725	100.7
TelosB without node `a95d`	2 759	26.6
Reports expected	3 787	—

Table 2: Number of reports per node by node type.
Report data is shown in Figure 13.

also confirm the correlation between hardware type and number of successful reports.

While the two hardware platforms share the same radio integrated circuit (IC), they use different antenna designs and different micro-controllers. Hypothetically, the difference in report success rates could stem from these software or hardware differences. For example the MicaZ antenna may be more effective and differences in micro-controller interrupt handling and latency may effect channel capture dynamics.

The above reporting method relies on the single-hop nature of the network to simplify the reporting of node data to the base station node. In order to test multi-hop algorithm performance, the above method was retained by utilizing the per-packet transmission power control present in TinyOS and the node radio hardware. In the multi-hop case, all synchronization algorithm traffic was transmitted at minimum power, which had the effect of allowing a multi-hop arrangement of nodes on a table-top scale. However, the base station interrogation packets and report packets were transmitted at full power, which allowed each node to be in direct contact with the base station node.

While similar to the single-hop case, this scheme does introduce additional complexity to the network dynamics since the clear channel assessment at each node can ascertain only if either a neighbouring node is transmitting synchronization traffic or if any node is transmitting report traffic, but remains deaf to synchronization traffic being transmitted by a non-neighbouring node. This allows for more collisions between synchronization traffic and reporting traffic since the propagation properties have been constructed to be highly non-symmetric in this case. However, since reporting traffic sees a single-hop network, the reporting traffic presents uniform network load at each node. A data collection strategy that presents uniform load at each node is preferable to tree-based multi-hop data collection strategies which might introduce bias by requiring nodes closer to the root of the collection tree to handle more network traffic.

5.4 Hardware Characterization

In order to understand the nature of the mote hardware and TinyOS systems software, clock characterization testing was conducted. Testing was performed using the collection methods detailed above. The clock values reported by the TinyOS system

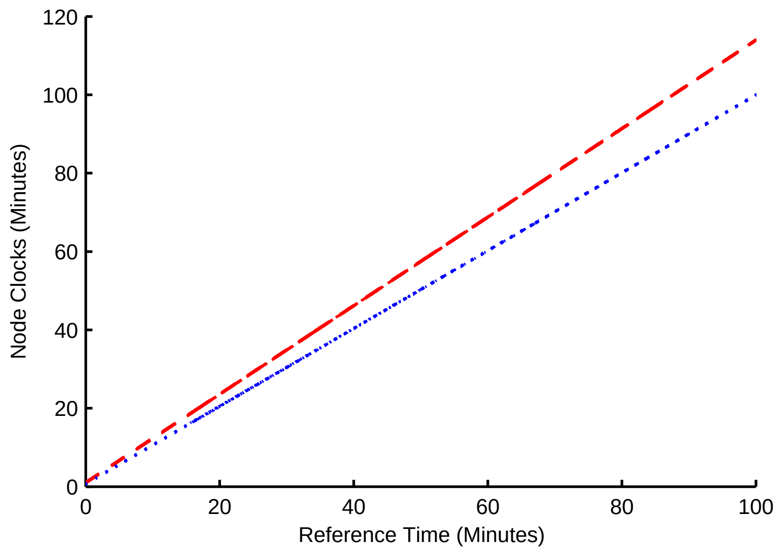


Figure 14: Uncorrected clocks showing 14 faster TelosB clocks and the 14 slower MicaZ clocks. All TelosB clocks fall within one line width of one-another, as do all MicaZ clocks. However, the two groups are clearly distinct.

were reported directly without correction. The base station used was a MicaZ node.

The clock processes of 28 nodes are shown in Figure 14. Immediately obvious is the difference in clock rates between MicaZ and TelosB nodes. Linear least-squares regression was used to estimate the rate and offset of each clock. In this test the offset values represent the different times at which nodes were reset before the test started. The results of the linear regression are shown in Table 3 with the average clock rates for the two node types shown in Table 4. All clocks are measured relative to the MicaZ base node clock.

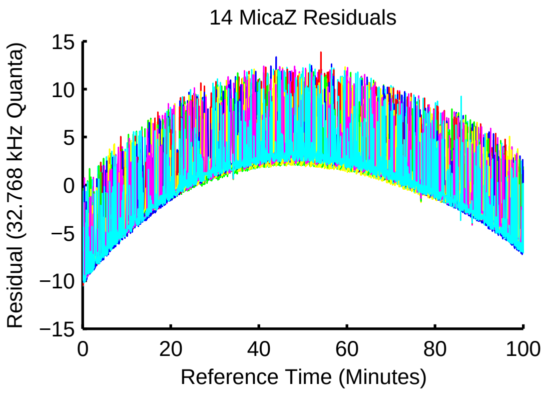
Taking the differences between the observed data and the clock fits we are left with the residuals for each fit. These residuals, for each fit, are shown in Figure 15. A single example of the residuals for each node type is shown in Figure 16. Again, the clock processes on the TelosB and MicaZ nodes are visibly different. The MicaZ residual graphs show a high frequency noise not visible in the TelosB graphs. The curvature of the residual graphs is also different, indicating that the higher-order

Relative Clock Rate	Node Type	Node Number
$+ 1.3754 \times 10^{-1}$	TelosB	22
$+ 1.3753 \times 10^{-1}$	TelosB	19
$+ 1.3752 \times 10^{-1}$	TelosB	11
$+ 1.3752 \times 10^{-1}$	TelosB	6
$+ 1.3752 \times 10^{-1}$	TelosB	13
$+ 1.3752 \times 10^{-1}$	TelosB	28
$+ 1.3752 \times 10^{-1}$	TelosB	9
$+ 1.3752 \times 10^{-1}$	TelosB	24
$+ 1.3752 \times 10^{-1}$	TelosB	18
$+ 1.3752 \times 10^{-1}$	TelosB	7
$+ 1.3752 \times 10^{-1}$	TelosB	3
$+ 1.3752 \times 10^{-1}$	TelosB	1
$+ 1.3751 \times 10^{-1}$	TelosB	5
$+ 1.3751 \times 10^{-1}$	TelosB	4
$+ 1.4846 \times 10^{-7}$	MicaZ	20
$- 2.0403 \times 10^{-7}$	MicaZ	26
$- 3.5208 \times 10^{-7}$	MicaZ	21
$- 4.5595 \times 10^{-7}$	MicaZ	25
$- 2.0590 \times 10^{-6}$	MicaZ	17
$- 2.0714 \times 10^{-6}$	MicaZ	2
$- 2.5867 \times 10^{-6}$	MicaZ	16
$- 3.8297 \times 10^{-6}$	MicaZ	15
$- 4.8487 \times 10^{-6}$	MicaZ	12
$- 5.0646 \times 10^{-6}$	MicaZ	8
$- 5.4962 \times 10^{-6}$	MicaZ	23
$- 6.6854 \times 10^{-6}$	MicaZ	27
$- 8.9559 \times 10^{-6}$	MicaZ	10
$- 9.4092 \times 10^{-6}$	MicaZ	14

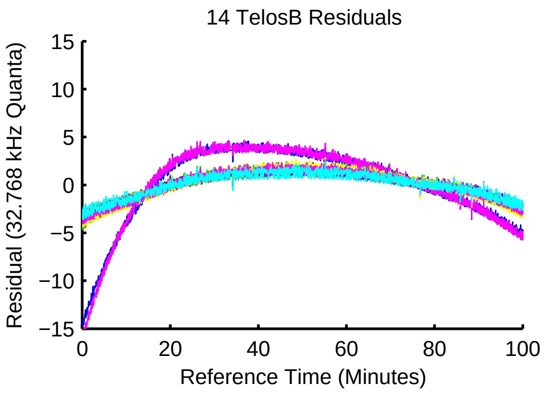
Table 3: Node clock rates relative to MicaZ base node clock rate.

Node Type	Average Clock Rate	Standard Deviation (σ)
TelosB	$+\ 1.3752 \times 10^{-1}$	6.9887×10^{-6}
MicaZ	$-\ 3.7050 \times 10^{-6}$	3.1733×10^{-6}

Table 4: Average clock rates relative to the MicaZ base node clock.

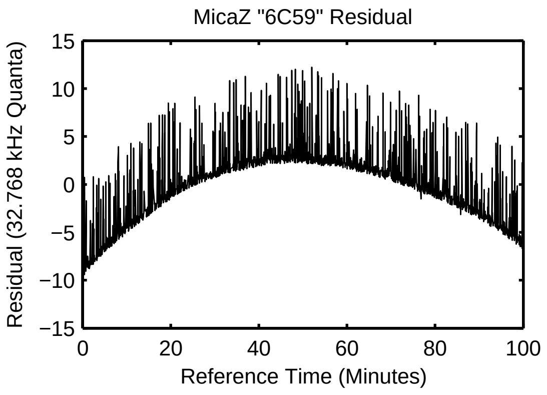


(a) MicaZ

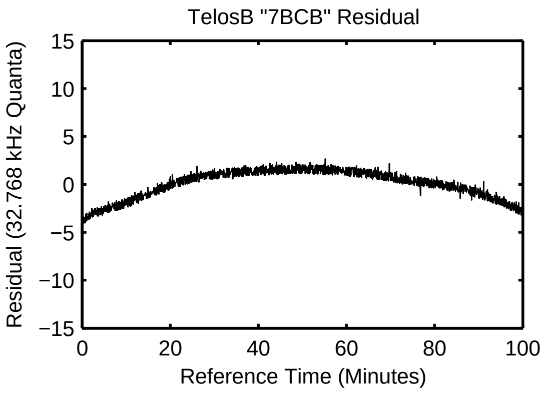


(b) TelosB

Figure 15: Residual plots for 14 MicaZ and 14 TelosB linear clock fits. Individual traces obscure each other.



(a) MicaZ



(b) TelosB

Figure 16: Residual plots for one TelosB and one MicaZ linear clock fit.

terms are different.

However, while higher-order terms are visibly present, the error from these is within ±15 quanta over 196×10^6 quanta. This non-linearity represents plus or minus 500 microseconds in 100 minutes or 76 parts per billion. At this scale, the level of non-linearity observed fits well with the use of a strictly linear clock model where continual on-line adaptation can absorb any errors introduced by non-linearity.

Furthermore, in Figure 15 we see that after approximately 20 minutes two TelosB node clocks exhibit a change in the shape of the residual plots. This indicates a change in the clock non-linearity. Because of the limited temporal resolution of the node clocks, combined with the small scale of the non-linearities, long periods would be required to estimate the non-linear terms in the clock model. However, if the non-linear terms exhibit instability over these scales, measurements of the non-linear terms becomes impossible. One possible explanation for the source of the instability is given below.

The grouping of the MicaZ and TelosB nodes in Figure 14 and Table 3 shows the importance of time synchronization in heterogeneous sensor networks. The average clocks for each hardware group are shown in Table 4. The standard deviation in each hardware group correspond to 95 percent confidence intervals of approximately ±14 ppm of the TelosB nodes and ±6.4 ppm for the MicaZ nodes. These confidence interval are in good agreement with the expected crystal accuracy of ±10 to ±20 ppm.[1]

The large difference between the MicaZ and TelosB clock appears to be generated in software, or more precisely, the way in which the system chooses to configure the available hardware counters. Instead of generating the 32.768 kHz node clock directly from the 32.768 kHz crystal, the TinyOS system software generates the 32.768 kHz

[1]This is the absolute accuracy of typical commercial watch crystals. For example, the various surface mount crystals manufactured by Seiko Instruments Inc. listed in [34] are available in ±10, ±20, and ±50 ppm specification.

clock from the processor clock.

As discussed in Section 2.2, the processor crystal for the MicaZ is a 7.3728 MHz crystal. While this frequency is exactly divisible by 32.768 kHz, the TinyOS system software divides this by 256 to generate a 28.8 kHz oscillator instead of a 32.768 kHz oscillator. If the TelosB clocks were exactly 32.768 kHz and the MicaZ base node clocks were exactly 28.8 kHz, this would correspond to the TelosB clock being 13.78 percent faster. From Table 4, the average of the TelosB clocks is 13.75 percent faster than the MicaZ base node clock.

On the TelosB hardware, the processor clock is not a crystal oscillator but a digitally controlled oscillator (DCO) implemented as a ring oscillator. The TinyOS system adjusts the tuning of this DCO to a nominal 4 MHz based on the 32.768 kHz oscillator. The system periodically checks for accumulated error between the 32.768 kHz crystal and 4 MHz processor oscillator and adjusts the DCO in an attempt to correct this error.

Overall, the clocks presented by the TinyOS system could be improved without hardware changes. However, the clocks have been shown to be linear within less than 2 ms over over a 100 minute period.

5.5 CS-MNS Performance

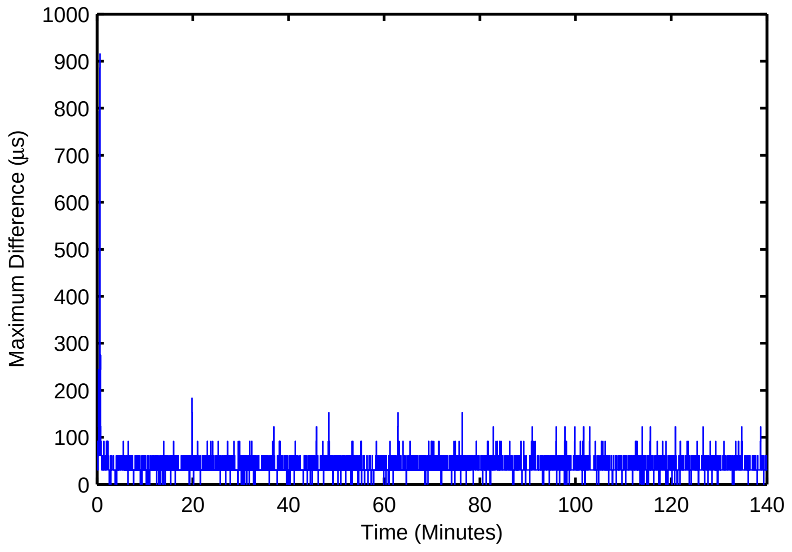
The performance of the CS-MNS implementation was tested using the methodology outlined in Section 5.3. Because initial synchronization is taken from the initial test-start message broadcast to all nodes, the accuracy of this initial synchronization is within approximately 30 μs, representing the granularity of 32.768 kHz clock used to time stamp the message.

However, in practise initial synchronization may be imprecise for large multi-hop groups. For some tests, in order to allow tests to be conducted that mirror multi-hop conditions with imprecise synchronization, a uniform pseudo-random value was added by each node to the epoch calculated at the intial synchronization.

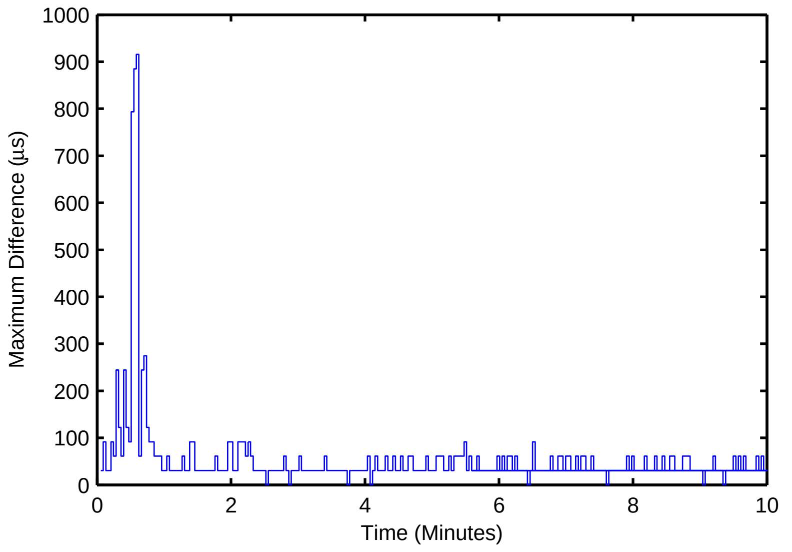
For the purposes of testing CS-MNS performance, each node broadcasts beacons with a Poisson process schedule. Unless otherwise noted, the average inter-beacon time at each node is 30 seconds. This results in a composite beacon rate for the group of $^{N}\!/_{30}\ ^{\text{beacons}}\!/_{\text{s}}$. As suggested in Section 3.4, the control gain was set at $k = 0.5$.

Shown in Figure 17 are the maximum clock differences observed among a group of 14 TelosB nodes in a single-hop group running the CS-MNS algorithm. The value of $\beta_{BIAS} = 20\,000$ ticks, combined with the close initial synchronization of approximately 30 μs, results in some initial divergence with a peak error of 915 μs occurring after approximately 35 seconds. This peak error corresponds to the error that would accumulate in 35 seconds between two clocks with a 26 ppm difference in rate, which is just outside of the uncorrected hardware tolerance. Indeed, inspection of the raw data indicates that the large error observed is due to node '669A'. During the first second of the test, node '669A' received one beacon successfully, but this beacon resulted in no change of the s_j correction factor. This node did not successfully receive any more beacons until after 35 seconds and thus the large error is due to the accumulated clock error at node '669A'. The next largest synchronization error observed is 275 μs. For the remainder of the test, clock granularity appears to be the primary factor limiting the CS-MNS algorithm, since the maximum observed clock differences are comparable to the quantization errors inherent in the beacons and the measurement.

In Figure 18 the clock adjustments made at each node are given for the same test shown in Figure 17. As all of the clock adjustments are negative, the average group clock rate is necessarily slower than the average of the hardware clock rates. Indeed, as all of the adjustments are negative, even the slowest hardware clock has been slowed by adjustment, which puts this clock rate outside of the range of clock



(a) Complete 140 minute experiment.



(b) Detail from first 10 minutes.

Figure 17: Synchronization errors for 14 TelosB nodes in a single-hop group running the CS-MNS algorithm ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks).

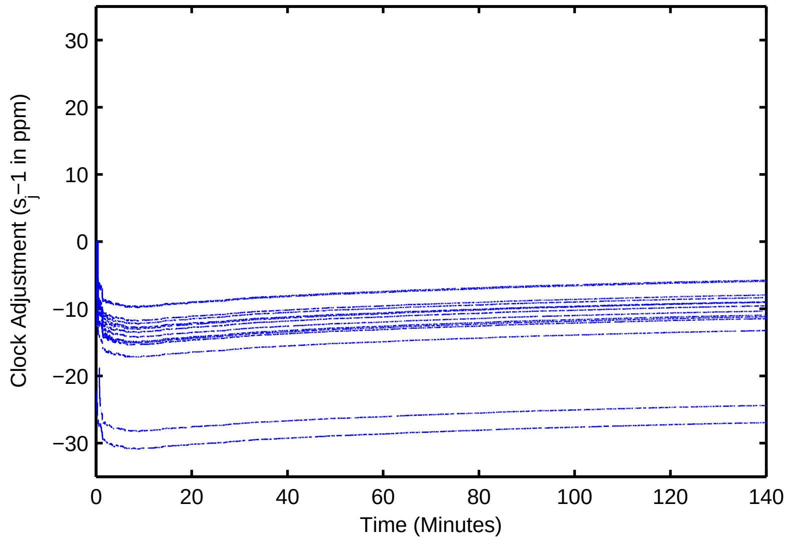


Figure 18: Adjustment factors for 14 TelosB nodes in a single-hop group running the CS-MNS algorithm ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks).

rates defined by the hardware clock rates. However, the relative relationship between the adjustment factors remains roughly constant for the period beyond 10 minutes, which agrees with the intuitive understanding that the CS-MNS algorithm makes large initial corrections to bring the clock rates into agreement, followed by smaller long-term corrections.

The experimental results shown for the 14 nodes group in Figure 17 compare well with the simulation results for a 30 node group shown in Figure 2(b). Since the experiment and simulation both use the same beacon rate at each node, the total beacon rate in simulation is approximately double that in experiment. The peak error is somewhat higher in experiment, but without the sample from node '669A' the next highest error of 275 μs is closer to the peak of the upper bound from simulation. However, slower convergence is expected in experiment as a result of the slower beacon rate and this, in turn, means that larger peak errors are expected. As expected, the convergence time is approximately 58 seconds in experiment while this time is only 25 seconds in simulation. Furthermore, the final synchronization

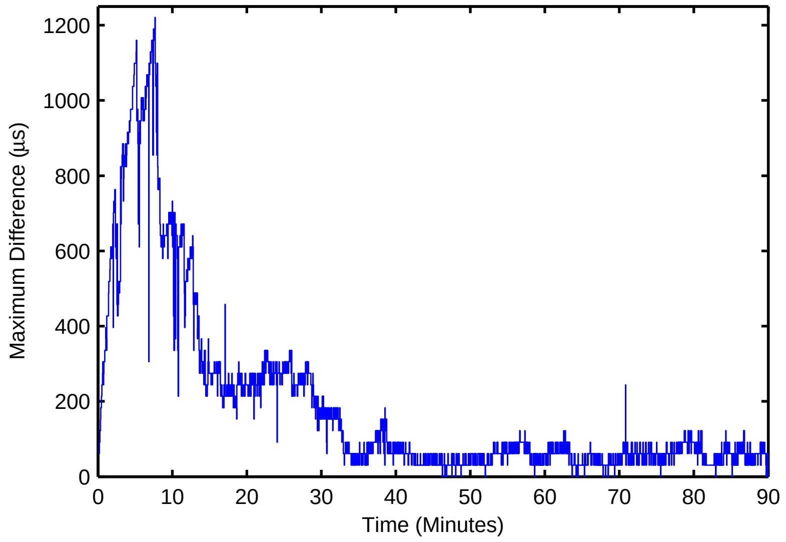


Figure 19: Synchronization error for 8 MicaZ nodes in a single-line multi-hop topology running the CS-MNS algorithm ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks).

error obtained in experiment is approximately 150% of the final error obtained in simulation. However, some allowance must be made for measurement error which is not present in the simulation results.

Shown in Figure 19 are the results observed from 8 TelosB nodes arranged in a multi-hop line. Each node can communicate only with its immediate neighbours which gives a network diameter of 7. The maximum observed error occurs after approximately 7.7 minutes and has a magnitude of 1.2 ms. This peak error corresponds to an accumulated error of approximately 2.6 ppm, which is an order of magnitude smaller than the hardware accuracy and shows that significant equalization of the clock rates has already occurred by this point.

Indeed, Figure 20 shows the correction factors applied at each nodes. Again, within the first 10 minutes the adjustment factors exhibit significant adaptation and have moved into a configuration close to their final configuration, with the relative ordering established. Comparing with the graph in Figure 19, the algorithm appears to settle between 30 and 40 minutes with minimal fine tuning of the correction factors

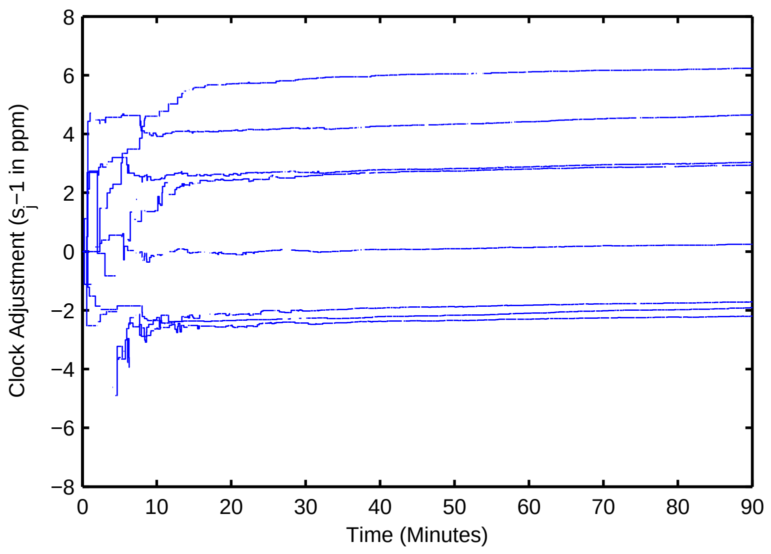


Figure 20: Adjustment factors for 8 MicaZ nodes in a single-line multi-hop topology running the CS-MNS algorithm ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks).

occurring after this point.

The shape of the graph in the multi-hop case is somewhat different than the shape seen in the single-hop case shown in Figure 17. Aside from the sustained, approximately-linear increase in error during the first 10 minutes of the test, there appear to be smooth bumps in the graph not obvious in the single-hop case. Two smooth bumps are visible around 60 minutes and another bump occurs around 80 minutes. The bumps are most likely a result of a slower reaction time in the multi-hop case. With the Poisson beacon transmission schedule used, the expected time for information to propagate through the 7 hops between the two most distant ends of the network is 3.5 minutes, whereas in the 14 node single-hop case this time is only 2.1 seconds, suggesting that the presence of longer-period structures would be reasonable in the multi-hop case.

Compared to the simulation results for this scenario, as shown in Figure 7, the testbed results are generally consistent. The peak synchronization error from experiment is 1.2 ms while the peak value from simulation is 4.3 ms but with a wide

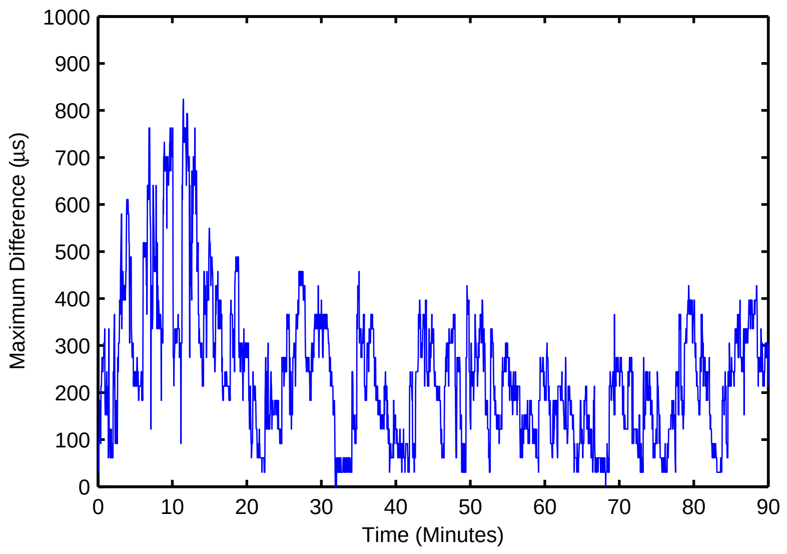


Figure 21: Synchronization error for 12 MICAz nodes arranged in a four-group multi-hop configuration. ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks)

confidence interval which contains the experimental value. Interestingly, the peak of the lower confidence interval from simulation is approximately 0.8 ms which compares well with the experimental peak of 1.2 ms despite these peaks occurring at different times. Comparing the experimental curve with the simulation curve shows that after approximately 40 minutes the rate of convergence slows significantly. However, in the case of simulation the convergence continues to obtain a final synchronization error about half of the final value obtained in experiment.

Another multi-hop network configuration was tested which consisted of a total of 12 MICAz nodes. The nodes were arranged in subgroups of three nodes where all nodes within a subgroup can communicate. The four subgroups were arranged in a line such that any node in each subgroup could communicate with the nodes in adjacent subgroups. The network topology is shown in Figure 8.

The synchronization error observed with this network arrangement is show in Figure 21. The synchronization error shows some initial divergence up to a peak value over 800 μs but within 20 minutes the error is less than 500 μs. The CS-MNS

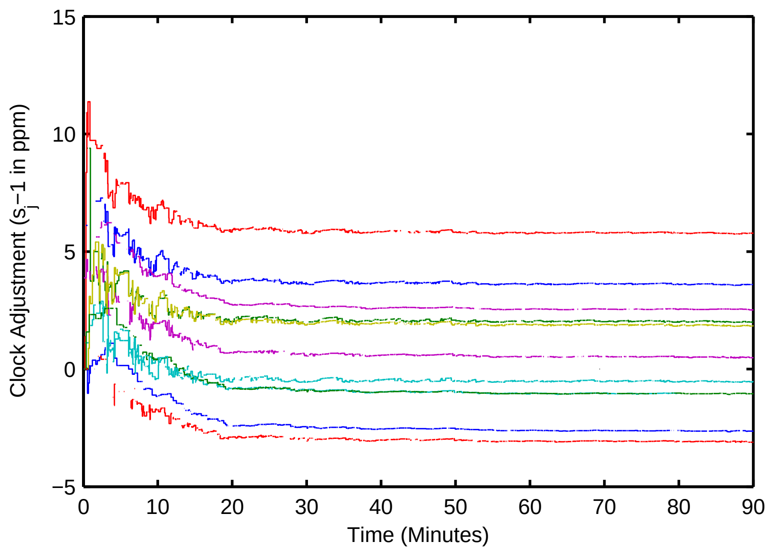
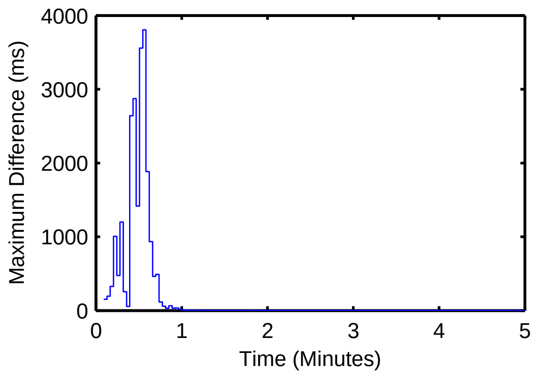


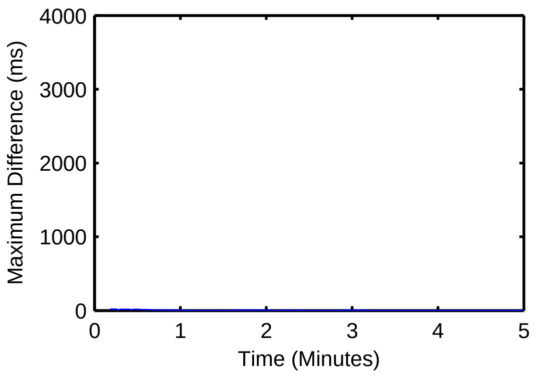
Figure 22: Adjustment factors for 12 MICAz nodes arranged in a four-group multi-hop configuration. ($k = 0.5$ and $\beta_{BIAS} = 20\,000$ ticks)

clock adjustments at each node are shown in Figure 22. While the adjustments appear to achieve a steady state, it is not clear from the synchronization error in Figure 21 that CS-MNS has achieved a steady state. Indeed, the results are significantly worse than the previous experimental results shown for the multi-hop line topology despite the increase in connectivity in the group topology. However, after 90 minutes an error of 40 ppm would be expected to accumulate an error of 216 ms, so even a 500 μs error after this time represents a significant improvement over uncorrected clocks.

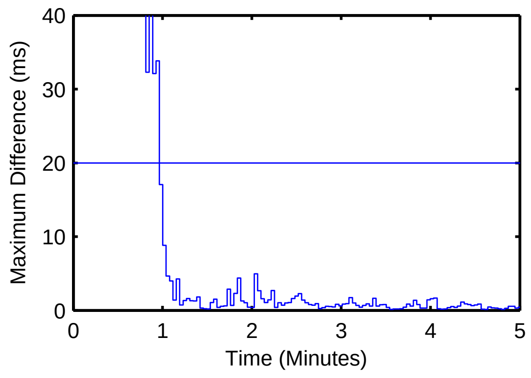
The experimental results in Figure 21 do not compare well with the simulation results in Figure 9. While the peak error observed in experiment was within the confidence bounds from simulation, this peak occurred significantly later in experiment. More importantly, while the simulation results show clear convergence after 200 seconds, the experimental results show only weak convergence after 20 minutes and do not show clear convergence even after 90 minutes. In the experimental setup, the links within each group are both shorter and are oriented favourably with regards to the shape of the MicaZ radiation pattern. This points to a possible unmodeled



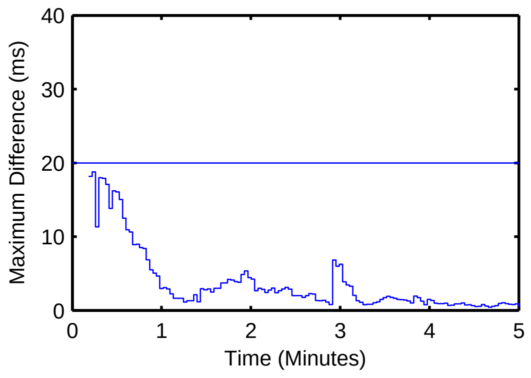
(a) $\beta_{BIAS} = 0$



(b) $\beta_{BIAS} = 13\,100\,000$ ticks



(c) $\beta_{BIAS} = 0$



(d) $\beta_{BIAS} = 13\,100\,000$ ticks

Figure 23: CS-MNS results from 14 MicaZ nodes with an artificial initial synchronization error of up to 20 ms. The network is arranged in a single-hop group. The horizontal line represents the initial 20 ms synchronization error. The same data is shown in the top and bottom figures at different scales.

bias effect where the links between groups have low signal to noise ratios and higher error rates while the links within each group have high signal to noise ratios and low error rates. In the simulation setup both types of links have zero error rate.

Shown in Figure 23 are the results from 14 TelosB nodes arranged in a single-hop topology with artificial initial synchronization error of up to 20 ms, running CS-MNS both with and without a bias term. Strong divergence is clearly visible in the case without bias term. The maximum disagreement between nodes peaks around 33 seconds into the run at almost 190 times the initial clock disagreement.

The rather large bias value used is suggested by Equation 24 with an initial offset error of 20 ms and a clock maximum rate error of 50 ppm. With the addition of this bias term, the maximum disagreement in the group tends downward during the initial minute of the test.

However, in both cases the CS-MNS algorithm serves to correct the initial error, with the results being comparable for the period after the first minute. Further, a rate error of 40 ppm would accumulate to 12 ms of error after 5 minutes. Thus the total disagreement of less than 1 ms observed after 5 minutes in both cases represents an improvement over the expected hardware clocks.

Not visible in Figure 23 is the effect that the initial divergence has on the overall group clock rate. Without a bias term, the average value of the correction factor s_j after 5 minutes is approximately 1.06, which is well outside the expected range of hardware clock rates. However, with the bias term the average value of the s_j adjustments is approximately 0.999998, which is 2 ppm less than unity and well within the expected range of hardware clock rates. Note that the same hardware was used in both cases.

Since the experimental parameters are different than those used in simulation shown in Figure 3, direct comparison is not possible. However, the overall effect of the bias term is consistent between the simulation and experiment.

The differences observed in group clock rates are consistent with the analytical presentation given earlier. In fact, the peak disagreement observed was 3.8 s, which occurred at $\tau = 33$ s and at this point the CS-MNS update law, given by Equation 2, with $k = 0.5$ would apply a correction of approximately 6%. However, once a large correction, resulting from a large clock difference, is applied, this correction will persist and at the end of the 5 minute test the average of the clock adjustments are still showing a 6% increase above nominal.

Technically, relative clock synchronization is achieved even if the aggregate group

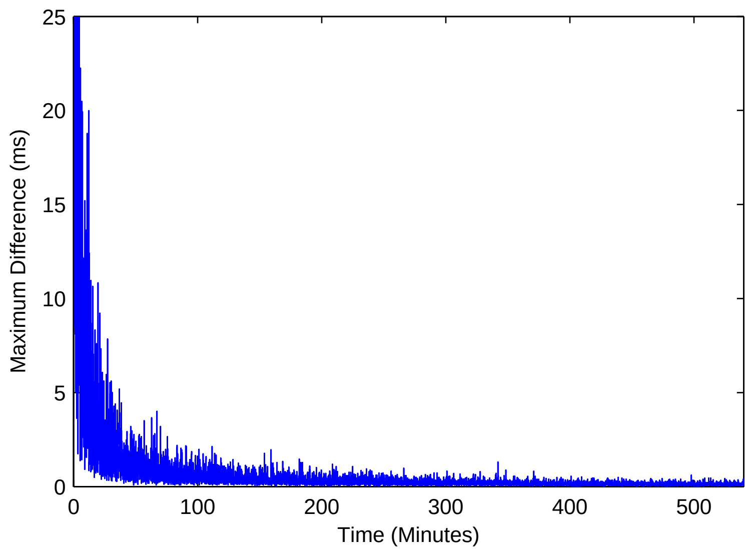


Figure 24: CS-MNS results for a mixed group of 7 TelosB and 7 MICAz nodes.

rate clock rate moves outside of the range defined by the initial clock rates, as was observed in the case without bias term. However, if the synchronized aggregate rate is hard to predict from the initial conditions, this can complicate a number of possible applications. For example, applications using the group clock to drive periodic operations will introduce a sensitivity to the unpredictably group clock on battery life, as periodic operations are repeated more often.

Large difference in group clock rate can also complicate applications requiring previously-independent groups of nodes to join and agree on a combined relative synchronization. In this case, the application designer has two choices. First, the joined network can be allowed to synchronize the clocks with rate differences larger than the hardware differences. Or second, methods to detect the joining of the groups can be added and the node clocks can be reset to their underlying hardware clocks, thus reducing the potential range of clock rates.

Figure 24 shows the observed clock errors for a group of 7 TelosB and 7 MicaZ nodes in a single-hop group. As shown in Section 5.4, the TelosB hardware clocks

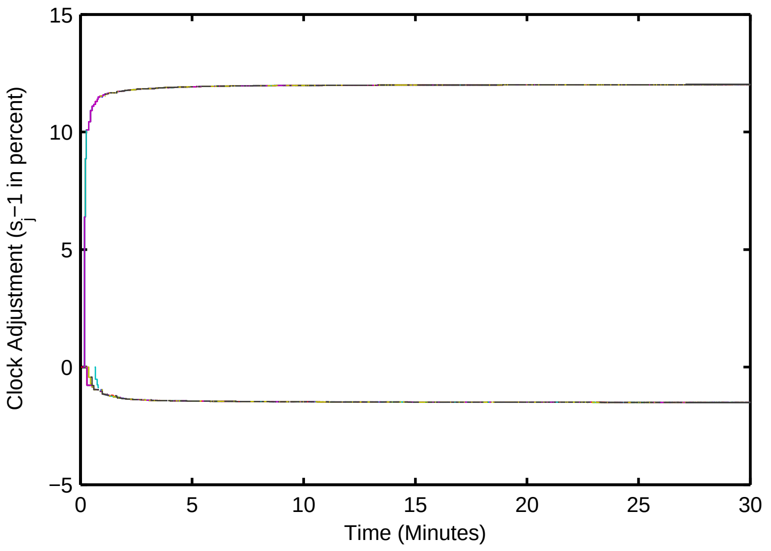


Figure 25: CS-MNS clock adjustments for a mixed group of 7 TelosB and 7 MICAz nodes. At this scale the thickness of the traces obscures the differences between nodes within each group.

are between 13 and 14 percent faster than the MicaZ hardware clocks. However, the test was conducted using a bias factor (β_{BIAS}) of 20 000 ticks as a network designer would be likely to determine an appropriate bias factor based only on the clock specifications. The observed synchronization errors are larger than in the case of homogeneous networks, with errors as high as 300 μs persisting after approximately 9 hours of run time. However, this represents a large improvement over the uncorrected clocks, which would have accumulated over an hour of error in this time.

The CS-MNS clock adjustments made by each node during the first 30 minutes of the mixed test are shown in Figure 25. The CS-MNS algorithm quickly begins to compensate for the large rate difference between the TelosB and MICAz nodes. The graph shows two distinct groups, with the upper trace consisting of slower MICAz nodes with increased adjustment factors and the lower trace consisting of TelosB nodes. The burden of reconciling the large clock differences between the nodes is shared between the two node types, although unevenly, with the TelosB nodes slowing

Topology	Synchronization Error		
	Maximum (μs)	Final (μs)	Time to Final (s)
Single-Hop	914[a]	31	58
Multi-Hop Group	824	250	1 920[b]
Multi-Hop Line	1 221	61	1600
Single-Hop w/ Clock Skew	1 055	220[c]	30 000[c]

[a] Single sample, next largest sample is 275 μs.
[b] Time to minimum error, point of convergence is not obvious.
[c] Very slow convergence appears to continue throughout the experiment.

Table 5: Summary of CS-MNS experimental results under different topologies. The final values column shows approximate centre values.

down to meet the MICAz nodes, which are speeding up. It should be noted that by introducing a large variation in the clock rate population we have introduced a large variation in the possible final group clock rates.

The simulation results shown in Figure 11 are markedly different from the experimental results shown in Figure 24. The simulation results indicate synchronization of only 45 μs after one minute while experimental results do not achieve this level of synchronization even after approximately 9 hours. No plausible explanation is obvious for the discrepancy between experiment and simulation.

5.6 CS-MNS Summary

The experimental results for CS-MNS under different topologies are summarized in Table 5. Additional experimental results confirm the observation of the improvement resulting from the addition of the bias term as predicted by analysis and simulation.

However, the steady state synchronization errors observed after convergence were higher in experiment than in simulation. Possible explanations for this include unmodeled sources of noise and unmodeled higher order clock terms. In particular, while the simulations do quantize the beacon transmissions, the simulations do not model noise caused by jitter between the node clocks.

5.7 FTSP Test Method

Both CS-MNS and FTSP produce beacons at a specified average frequency. In the CS-MNS tests above the beacons are randomly timed, while in FTSP the beacons have regular, periodic timing. However, choosing equivalent beacons rates for a fair comparison of the two algorithms must be done with some care. One of the desirable properties of CS-MNS is that whenever a node receives a beacon it may use this beacon to make an adjustment regardless of the node sending the beacon or the previous protocol state.

In contrast to CS-MNS, FTSP transmits many beacons that are not used. After an FTSP node receives a beacon, the node re-broadcasts the beacon in order to flood the synchronization data to lower tiers of the network tree. However, if a node receives a beacon from a node on the same or lower tier, the node ignores this beacon. In some network topologies this leads to many beacons that do not result in any clock adjustments.

For example, in the case of a single-hop network with 16 nodes, one master node will send a beacon which will be re-broadcast by 15 nodes. This results in a total of 16 message transmissions per beacon period. However, any node receiving a re-broadcast beacon will simply ignore the re-broadcast beacon in this topology since the re-broadcast beacon must necessarily originate from an equal or lower tier node.

Both the CS-MNS tests conducted here and the FTSP algorithm share beacon transmission duties uniformly among all nodes. Thus, in order to set comparable CS-MNS and FTSP beacon rates, the total number of message transmissions in the network were set equal. This approach was taken because the message transmissions have an associated energy cost and setting the total number of beacon transmissions in the network equal for the two algorithms sets the energy budget for the two algorithms approximately equal. This choice also sets the total spectrum usage approximately equal between the two algorithms, with beacon length being the only uncontrolled spectrum usage parameter.

Assuming that all nodes are reachable through flooding, setting the FTSP root beacon rate equation to the per-node CS-MNS beacon rate results in the same total number of transmissions under each algorithm. Since each FTSP node re-broadcasts received messages, the total message rate is equal to N times the root rate. Similarly, the total transmission rate for CS-MNS is N times the individual node beacon rate.

5.8 FTSP Performance

An example FTSP run is shown in Figure 26, which shows the maximum observed difference in clocks between 14 TelosB nodes in a single-hop configuration. The topmost axis includes all node observations, whereas the middle axis ignores data from nodes that report an invalid synchronization state. The lower graph shows the total number of reports received as well as the number of nodes that reported an invalid synchronization state.

After approximately 2 minutes the FTSP algorithm successfully synchronizes the nodes and holds this synchronization to within approximately 100 μs until about 55 minutes into the test. A large error then appears with a magnitude of approximately 17 hours. This error persists for the next 15 minutes. Interestingly, for the first 10 minutes that this large error is present, all nodes continue to indicate valid synchronization states. After this point, a single node begins to report an invalid synchronization state and the error among the remaining nodes with valid synchronization states drops to less than 200 μs. This implies that the time synchronization

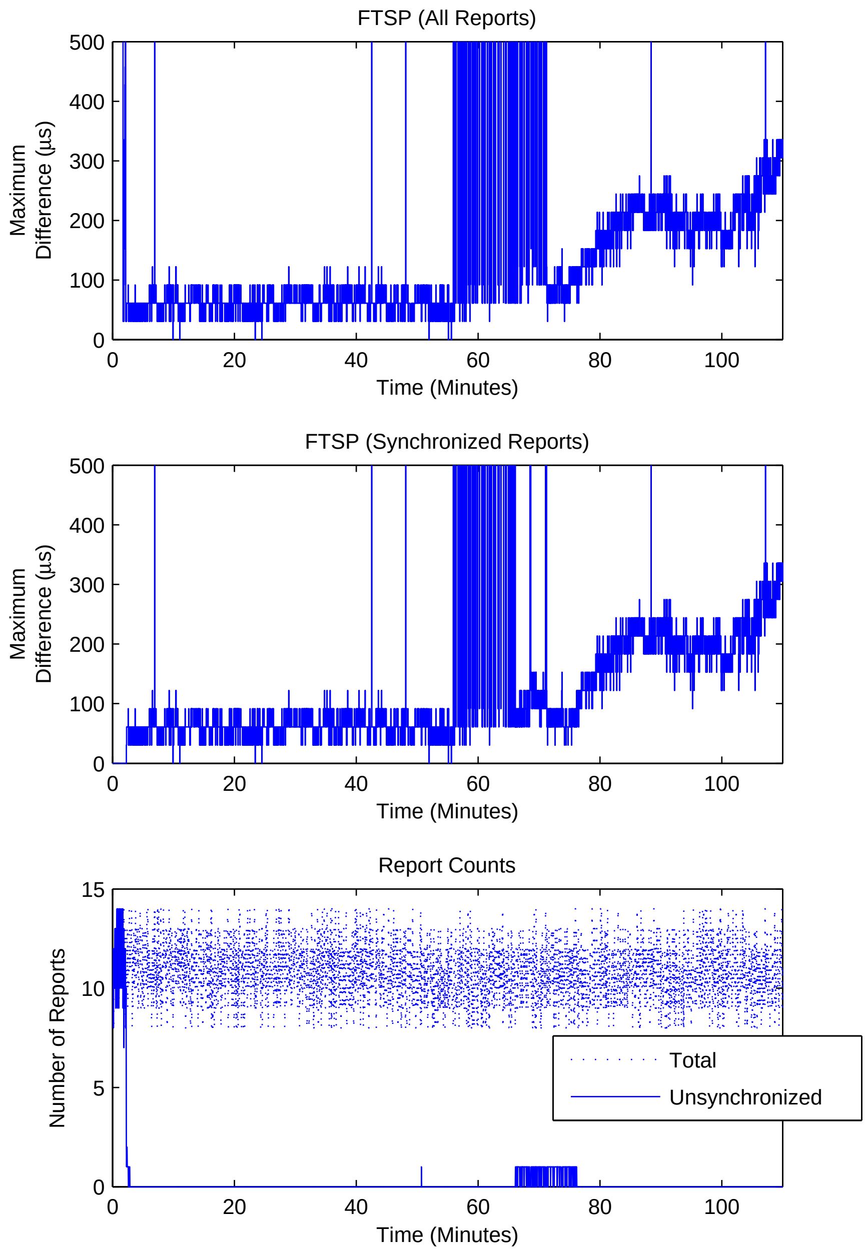


Figure 26: FTSP results for 14 TelosB nodes in a single-hop group.

of a single node has become corrupted and that it takes the FTSP algorithm 10 minutes (representing 20 rounds at the 30 second beacon period) to detect the problem and a further 5 minutes (10 rounds) to correct the synchronization.

However, within 5 minutes of the end of the large error, the group synchronization appears to grow progressively worse with the error climbing to 300 μs by the end of the run. One possible explanation for this behaviour would be that the misbehaving node has not correctly re-synchronized with the group.

The multi-hop performance of FTSP was tested using 8 MicaZ nodes arranged in a line topology. Each node can communicate only with its immediate neighbours resulting in a network diameter of 7 hops. The results from this test are shown in Figure 27. Immediately evident is the scale of the synchronization errors. There is an initial linear increase in the synchronization error which peaks with an error of 318 minutes after only 21 minutes of the test has elapsed. This accumulation of error represents a clock rate error of 1 500%. Obviously, with an error this large FTSP has a detrimental effect on synchronization during the initial 21 minutes.

Examining Figure 27 further, the FTSP algorithm appears to become more effective after the 27 minute mark, especially when only those nodes reporting a valid synchronization state are considered. The error among the synchronized nodes remains under 10 minutes for the rest of the test. After 45 minutes the error drops further to less than 2.5 seconds. However, this error is significantly worse than the error of 100 μs seen in the single-hop case.

Finally, it is interesting to note that during the 90 minutes of the test the longest period in which all of the nodes reported valid synchronization was approximately 2 minutes. In contrast, the single-hop test shows all nodes reporting valid synchronization for the majority of the test period. It appears that this multi-hop configuration is outside of the set of configurations in which FTSP is effective. This is significant because a reasonable prediction would be that the FTSP algorithm would successfully

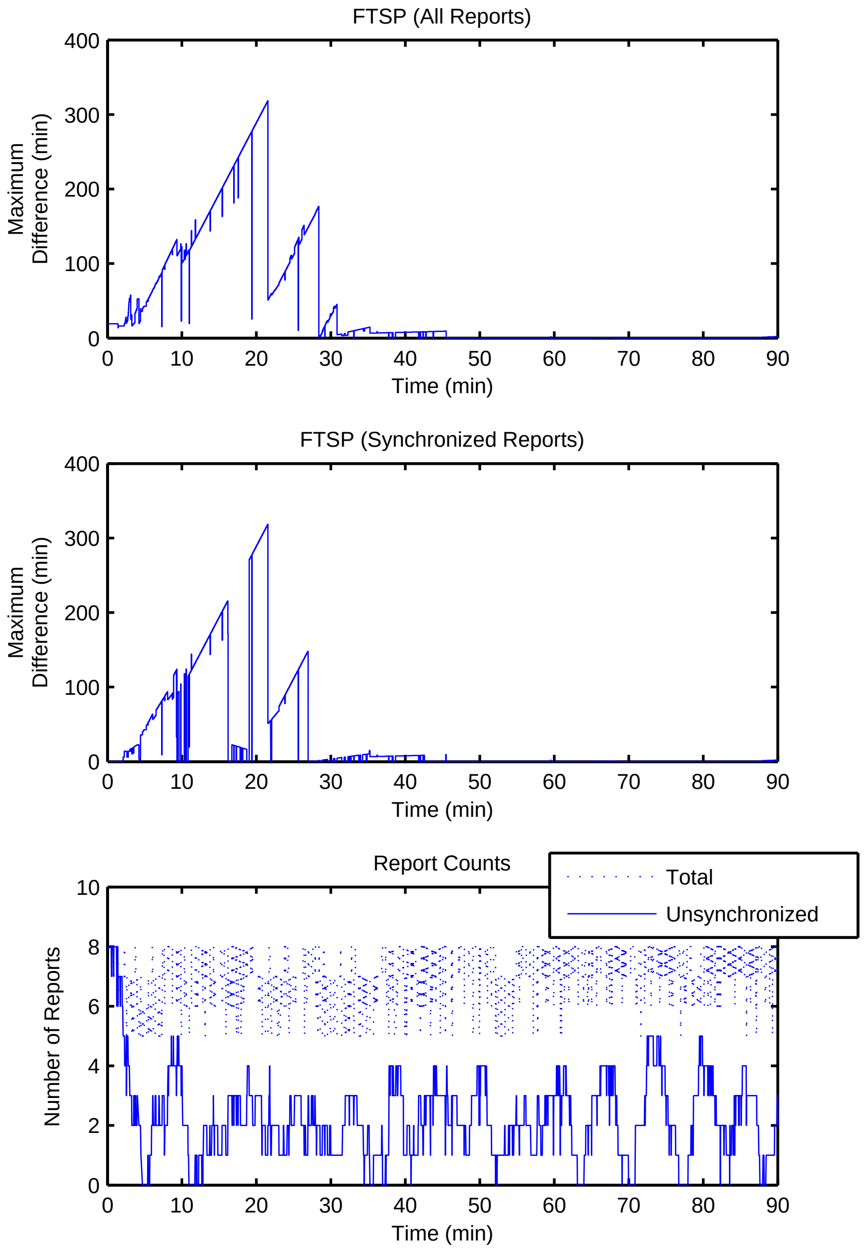


Figure 27: FTSP results from 8 MicaZ nodes in a multi-hop line configuration. Note that the vertical time axis is shown in minutes.

flood the synchronization data throughout this network.

A less extreme network configuration was tested consisting of a total of 12 MICAz nodes. The nodes were arranged in subgroups of three nodes where all nodes within a subgroup can communicate. The four subgroups were arranged in a line such that any node in each subgroup could communicate with the nodes in adjacent subgroups. This network topology is shown in Figure 8. The node with the lowest node-number, which FTSP will select as the master, was located in an end group. The FTSP results for this configuration are shown in Figure 28. The results for this configuration are much better than the results for a single line topology shown in Figure 27.

However, the subgroup configuration results in Figure 28 have synchronization errors measured in tens of seconds throughout the hour of the test. This is much larger than the 144 ms error that would be expected after an hour with an uncorrected error of 40 ppm. Additionally, the number of nodes reporting an invalid synchronization state remains high throughout the test.

The observations collected for FTSP in a single-hop environment with heterogeneous hardware are shown in Figure 29. While the FTSP algorithm does synchronize the node clocks initially, the algorithm repeatedly looses this synchronization throughout the test. The performance degrades to around 10 s of error approximately every 4 minutes when the algorithm appears to re-synchronize the nodes anew. Synchronized with, but out of phase with, the periodic degradation of synchronization is a periodic oscillation in the number of nodes reporting a valid synchronization state.

5.9 FTSP Summary

The experimental results for FTSP under different topologies are summarized in Table 6. The poor performance of FTSP in multi-hop topologies is not entirely consistent with the performance expected based on previously published results. For example,

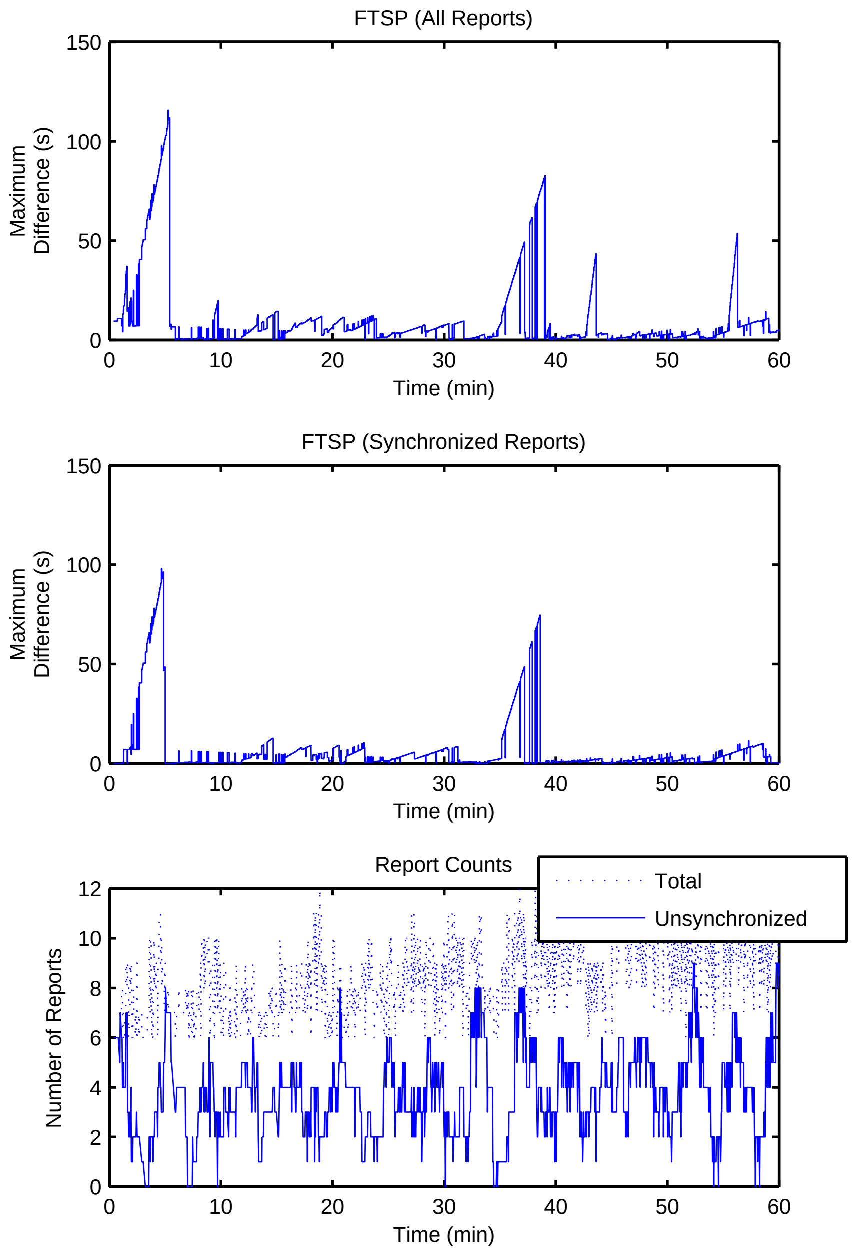


Figure 28: FTSP results for 12 MicaZ nodes arranged in 4 subgroups.

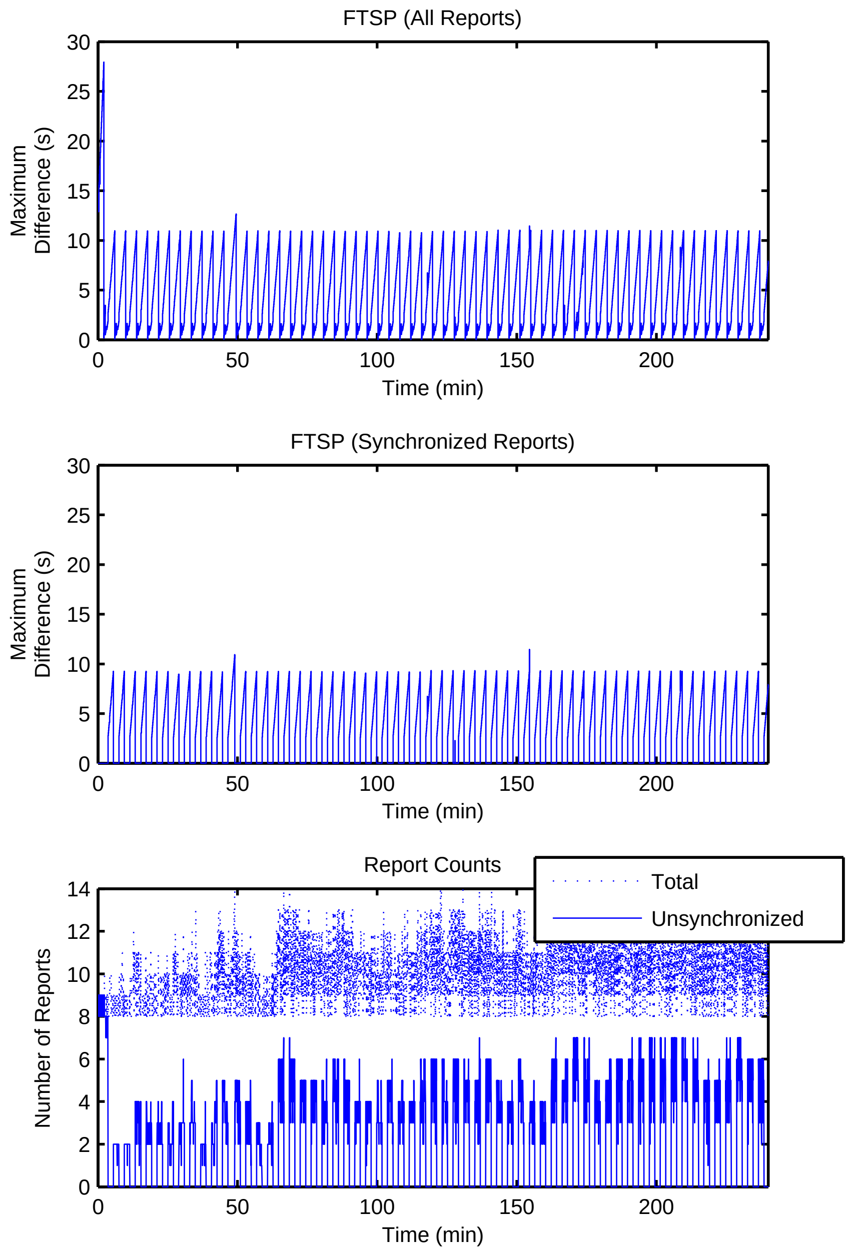


Figure 29: FTSP results for 7 MicaZ and 7 TelosB nodes in a single-hop configuration.

Topology	Synchronization Error		Time to Synchronize
	Maximum[a]	Final	
Single-Hop	335 μs [b]	61 μs[c]	2 min
Single-Hop w/ Clock Skew	12 s	5 s[d]	2 min
Multi-Hop Group	115 s	5 s	6 min
Multi-Hop Line	318 min	3 min[e]	5 min[f]

[a] Errors recorded prior to the initial point where all nodes report a synchronized state are excluded from the reported maximum.
[b] Value ignoring numerous outlier points and the greater than one hour errors observed during the period from 55 to 71 minutes.
[c] Midpoint error is given after 55 minutes. Synchronization degrades during the remainder of the experiment.
[d] Midpoint is given with peaks up to 11 s errors still present after 250 minutes.
[e] Significant portions of the experimental periods show an error of less than 2 seconds.
[f] Large errors are present despite all nodes reporting synchronization. Error is reduced after approximately 30 minutes.

Table 6: Summary of FTSP experimental results under different topologies.

in [29] the authors show testbed results for 60 Mica2 nodes in a multi hop grid topology. These results show synchronization errors ranging from a maximum of 15 μs during some periods to a maximum of 60 μs in other periods.

However, there are a number of important differences between the experiments in [29] and the test conducted here that may explain the difference in performance. The node hardware and operating systems are different and the FTSP implementations are different. In the current work, radio interference was also uncontrolled besides being a fairly typical office environment. The background radio environment is unspecified in [29].

More significantly, in [29] the network topology is enforced in software with the physical network topology being a single-hop network. This is very different from the results shown here, where the FTSP network traffic is sent at low power with only data collection occurring at full transmission power. This results in a physical multi-hop network topology but also reduces the signal to noise ratio for FTSP traffic.

Thus, presumably, resulting in higher error rates for FTSP traffic.

In [30] the authors report that when FTSP was tested by enforcing a 20 hop line network topology in software on Mica2 nodes the un-modified TinyOS 2.x implementation "failed to synchronize all nodes in the network even after a long time period". The authors in [30] modified FTSP to remove the root node selection process and to prevent FTSP from attempting to detect outlier samples. The authors report that these modifications allowed FTSP to synchronize a 20 hop line network. However, removing the root node selection process obviously prevents FTSP from both recovering from a failure of the root node and from handling ad-hoc networks, two desirable properties of FTSP.

In both the multi-hop single-line and multi-hop group topologies (Figures 27 and 28), as well as the heterogeous single-hop topology (Figure 29), the number of nodes reporting a synchronized state oscillates throughout the test. Only in the single-hop case (Figure 26) are the FTSP results free from this oscillation in number of synchronized nodes. Thess result hints that the FTSP algorithm is not completely stable in the multi-hop and heterogeneous topologies.

5.10 Comparison of CS-MNS and FTSP

A summary comparing the experimental synchronization achieved and time to synchronization for CS-MNS and FTSP is shown in Table 7.

In the homogeneous single-hop case, both CS-MNS and FTSP synchronize the network to better than 100 μs. Both algorithms achieve this synchronization withing the first two minutes of run time. However, after approximately one hour the FTSP algorithm appears to loose synchronization. The reasons for this sudden loss of synchronization and subsequent increase in errors shown in Figure 26 is not clear. CS-MNS, on the other hand, is free from any such errors.

Topology	CS-MNS		FTSP	
	Final Error	Sync. Time	Final Error	Sync. Time
Single-Hop	31 μs	58 s	61 μs[a]	2 min
Single-Hop w/ Clock Skew	220 μs[b]	500 min[b]	5 s[c]	2 min
Multi-Hop Line	61 μs	27 min	3 min[d]	5 min[e]
Multi-Hop Group	250 μs	32 min[f]	5 s	6 min

[a] Midpoint error is given after 55 minutes. Synchronization degrades during the remainder of the experiment.
[b] Very slow convergence appears to continue throughout the experiment.
[c] Midpoint is given with peaks up to 11 s errors still present after 250 minutes.
[d] Significant portions of the experimental periods show an error of less than 2 seconds.
[e] Large errors are present despite all nodes reporting synchronization. Error is reduced after approximately 30 minutes.
[f] Time to minimum error, point of convergence is not obvious. However, maximum error is low at 500 μs.

Table 7: Summary of experimental results for CS-MNS and FTSP under different topologies.

In the heterogeneous single-hop case both FTSP and CS-MNS perform poorly. The two algorithms show different types of errors in this case. However, the final synchronization error observed for CS-MNS is significantly lower than the error observed for FTSP. On the other hand, FTSP brought the group synchronization together in a mater of minutes instead of hours. The fact that the CS-MNS experimental results do not match the simulation results for this case suggest that there are more effects than the clock rate difference. In order to isolate these effects further experiments would be required. Potential experiments could introduce large clock rate differences in an otherwise homogeneous hardware environment. Alternatively, the large rate difference could be corrected and the experiments repeated, maintaining the heterogeneous hardware environment.

The disappointing experimental performance of FTSP in the multi-hop line topology with 7 hops may indicate that this topology is outside of the set of topologies where FTSP is effective. CS-MNS, on the other hand, performed fairly well in this topology. While convergence of CS-MNS was slow under this topology, compared to the single-hop topology, the time to converge was comparable to the time required for FTSP to reduce the large initial errors. However, CS-MNS achieved synchronization with an error of approximately 60 μs whereas FTSP achieved an error on the order of 2.5 seconds. Obviously, in this case CS-MNS outperformed FTSP on the same hardware with the same total communications cost.

In the multi-hop group topology FTSP performance was somewhat better than the FTSP performance in the multi-hop line case. Conversely, the CS-MNS performance was worse in the multi-hop group case as compared to the CS-MNS performance in the multi-hop line topology. While CS-MNS did not show clear convergence, the total error did not go above 500 μs during the test. The largest error during the CS-MNS experiment was smaller than the final FTSP synchronization value. Thus, even in this topology, the CS-MNS algorithm again achieved tighter synchronization than FTSP.

5.11 Summary

A simple characterization of the TinyOS 2.1 system and the MICAz and TelosB hardware led to the somewhat surprising result that the MICAz nominal 32.768 kHz clock is implemented as a 28.8 kHz clock. Analysis of the TinyOS code shows that this discrepancy is rooted in software. The clocks for both the MICAz and TelosB systems were found to be within one part-per-million of linear over a 100 minute period.

The behaviour of CS-MNS was found to agree qualitatively with simulation except in the case of the heterogeneous hardware environment. In particular, the effect of the bias term was consistent with simulation. The experimental results show an increase in synchronization error over the simulation results. This increase in error is most likely a result of unmodeled noise sources. Further testing would be required in order to isolate the factors causing the discrepancy between the CS-MNS simulation and experimental results in the heterogeneous hardware environment.

The CS-MNS and FTSP algorithms exhibited qualitatively different performance. The FTSP algorithm was found to perform more poorly than expected. In particular, the tendency of the FTSP algorithm to exhibit a mix of synchronized and unsynchronized periods was particularly damaging to the overall evaluation of FTSP performance since the CS-MNS results were free from this.

Under conditions of equal communication costs, in all experiments CS-MNS outperformed FTSP in terms of final synchronization error. However, in most conditions, the synchronization errors under CS-MNS took longer to converge to a final value as compared to FTSP. But, under all conditions where CS-MNS converged more slowly than FTSP, the maximum synchronization error under CS-MNS was smaller than the final error under FTSP. Overall, the experimental performance of CS-MNS was promising in comparison to the performance of FTSP.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Motivated by a further understanding of dynamics of the clock sampling mutual network synchronization (CS-MNS) algorithm, we have presented an analysis of the CS-MNS algorithm. This analysis emits a proof of convergence in the absence of initial offset errors and provides bounds, applicable in the general case, outside which the CS-MNS algorithm continues convergence. The understand provided by this analysis also led to the modification of CS-MNS through the addition of a bias term to control initial divergence.

Further exploration of the dynamic behaviour of CS-MNS is provided through numerical simulation. The numerical simulation was used to investigate the performance of CS-MNS in networks with varying topologies. The simulations also serve to show the improvement in performance offered by the modification of the CS-MNS algorithm through the addition of the bias term.

The CS-MNS algorithm was implemented, and its the performance was measured experimentally, in a wireless sensor network (WSN) testbed. The testbed results agree qualitatively with the presented numerical simulation results as well as previously published simulation results. Various topologies were tested. The effectiveness of the additional bias term is was also shown through experimental measurement.

The flooding time synchronization protocol (FTSP) algorithm, as implemented in TinyOS 2, was tested experimentally in the same testbed in networks with various topologies. The performance of CS-MNS is shown to be generally good in comparison to FTSP, although the FTSP performance was below expectation, particularly in multi hop topologies. CS-MNS convergence was often slower than FTSP. But even in these cases CS-MNS achieved tighter synchronization. Qualitatively, FTSP and CS-MNS algorithms exhibited different dynamics.

6.2 Future Work

In the case of heterogeneous hardware, the testbed results show a decrease in synchronization performance larger than the decrease expected from simulation. Further experiments may be able to isolate the root cause of this decrease in performance. Possible further experiments could correct the differences between the hardware platform clocks in software, or further experiments could introduce a large clock rate error, in software, tested on a network of identical hardware.

Future work which performed further experimental evaluation of the CS-MNS algorithm where CS-MNS is used to synchronize higher frequency clocks may provide further insight into the behaviour of CS-MNS after initial convergence.

The testbed would benefit from the addition of a reliable, separate channel that allowed reporting of test data without adding traffic to the wireless channel used by the nodes under test. Additionally, a method to distribute a shared clock or timing pulses to each node would allow more accurate measurement of the time process at each node.

Previous work shows promising simulation results for CS-MNS in networks with mobility. Future experiments could evaluate the performance of CS-MNS in a testbed which included node mobility. Additionally, there remains significant related theoretical, simulation, and experimental work that can be done on the behaviour of CS-MNS when networks are segregated, when networks are joined, and when nodes are added and removed from networks. Similarily, there remains possible analytical and experimental work toward identification of an optimal value for the control gain k, as well as investigation into the effects of the k value on group dynamics.

As presented to date, no security provisions are made in the design of CS-MNS. If the correctness of the WSN operation depends on the validity of the time synchronization data, synchronization protocols could become a viable target for an attacker. Appendix B give a brief discussion of some of the challenges and some ideas applicable towards a secure version of the CS-MNS algorithm.

List of References

[1] K. Bult, A. Burstein, D. Chang, M. Dong, M. Fielding, E. Kruglick, J. Ho, F. Lin, T. Lin, W. Kaiser, *et al.*, "Low power systems for wireless microsensors," in *International Symposium on Low Power Electronics and Design, 1996.*, pp. 17–21, 1996.

[2] G. Pottie, "Wireless sensor networks," in *Information Theory Workshop, 1998*, pp. 139–140, 1998.

[3] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, pp. 281–323, May 2005.

[4] C. Rentel and T. Kunz, "A clock-sampling mutual network time-synchronization algorithm for wireless ad hoc networks," *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 1, pp. 638–644 Vol. 1, March 2005.

[5] C. H. Rentel and T. Kunz, "Clock-sampling mutual network synchronization for mobile multi-hop wireless ad hoc networks," *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pp. 1–7, Oct. 2007.

[6] C. H. Rentel and T. Kunz, "A mutual network synchronization method for wireless ad hoc and sensor networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 5, pp. 633–646, May 2008.

[7] E. McKnight-MacNeil and T. Kunz, "Behavior of clock-sampling mutual network synchronization in wireless sensor networks," in *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*, pp. 633–638, ACM, 2009.

[8] E. McKnight-MacNeil, C. Rentel, and T. Kunz, "Behavior of clock-sampling mutual network synchronization in wireless sensor networks: convergence and security," *Wireless Communications and Mobile Computing*, vol. 10, no. 1, pp. 158–170, 2010.

[9] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, and M. Gouda, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks*, vol. 46, no. 5, pp. 605–634, 2004.

[10] D. Doolin and N. Sitar, "Wireless sensors for wildfire monitoring," in *Proceedings of SPIE*, vol. 5765, p. 477, 2005.

[11] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pp. 88–97, ACM, 2002.

[12] F. Osterlind, E. Pramsten, D. Roberthson, J. Eriksson, N. Finne, and T. Voigt, "Integrating building automation systems and wireless sensor networks," in *12th IEEE Conference on Emerging Technologies and Factory Automation, Patras, Greece*, Citeseer, 2007.

[13] Memsic Corporation, "MICAz datasheet." `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz`.

[14] Memsic Corporation, "TelosB datasheet." `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb`.

[15] Texas Instruments, "CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Datasheet." `http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=cc2420&fileType=pdf`.

[16] Atmel Corporation, "ATmega128(L) Datasheet." `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

[17] Texas Instruments, "MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (Rev. F) Datasheet." `http://www.ti.com/lit/gpn/msp430f1611`.

[18] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, *et al.*, "Tinyos: An operating system for sensor networks," *Ambient Intelligence*, pp. 115–148, 2005.

[19] P. Levis and D. D. E. Gay, "TinyOS programming," 2006.

[20] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[21] Q. Wang, W. Chen, R. Zheng, K. Lee, and L. Sha, “Acoustic target tracking using tiny wireless sensor devices,” in *Proceedings of the 2nd international conference on Information processing in sensor networks*, pp. 642–657, Springer-Verlag, 2003.

[22] J. Acebrón, L. Bonilla, C. Pérez Vicente, F. Ritort, and R. Spigler, “The Kuramoto model: A simple paradigm for synchronization phenomena,” *Reviews of modern physics*, vol. 77, no. 1, pp. 137–185, 2005.

[23] D. Mills, “Internet time synchronization: The network time protocol,” *IEEE transactions on communications*, vol. 39, pp. 1482–1493, 1991.

[24] F. Ren, C. Lin, and F. Liu, “Self-correcting time synchronization using reference broadcast in wireless sensor network,” *IEEE Wireless Communications Magazine*, vol. 15, pp. 79–85, Aug. 2008.

[25] L. Huang and T. Lai, “On the scalability of IEEE 802.11 ad hoc networks,” in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, p. 182, ACM, 2002.

[26] D. Zhou, L. Huang, and T. H. Lai, “On the scalability of IEEE 802.11 ad-hoc-mode timing synchronization function,” *Wireless Networks*, vol. 14, no. 4, pp. 479–499, 2008.

[27] D. Zhou, “A Compatible and Scalable Clock Synchronization Protocol in IEEE 802.11 Ad Hoc Networks,” in *Proceedings of the 2005 International Conference on Parallel Processing*, p. 302, IEEE Computer Society, 2005.

[28] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, “Robust multi-hop time synchronization in sensor networks,” in *Proceedings of the International Conference on Wireless Networks (ICWN’04)*, pp. 454–460, 2004.

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

[30] C. Lenzen, P. Sommer, and R. Wattenhofer, “Optimal clock synchronization in networks,” in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pp. 225–238, ACM, 2009.

[31] C. H. Rentel, *Network Time Synchronization and Code-based Scheduling for Wireless Ad Hoc Networks*. PhD thesis, Carleton University, January 2006.

[32] N. Freris and P. Kumar, "Fundamental limits on synchronization of affine clocks in networks," *Decision and Control, 2007 46th IEEE Conference on*, pp. 921–926, Dec. 2007.

[33] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on Automatic Control*, vol. 50, pp. 169–182, Feb. 2005.

[34] Seiko Instruments Inc., "SII Quartz Crystals Product Catalogue 2008-2009." `http://www.sii.co.jp/compo/catalog/crystal_en.pdf`.

[35] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, D. Culler, and D. Gay, "Tinyos enhancement proposal: Hardware abstraction architecture." `http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/doc/html/tep2.html`.

[36] C. Sharp, M. Turon, and D. Gay, "Tinyos enhancement proposal: Timers." `http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/doc/html/tep102.html`.

[37] Maxim Integrated Products, "DS2401 silicon serial number datasheet." `http://datasheets.maxim-ic.com/en/ds/DS2401.pdf`.

[38] Maxim Integrated Products, "DS2411 silicon serial number with V_{CC} input datasheet." `http://datasheets.maxim-ic.com/en/ds/DS2411.pdf`.

[39] Maxim Integrated Products, "Application note 126: 1-Wire communication through software." `http://www.maxim-ic.com/app-notes/index.mvp/id/126`.

[40] D. Xiaojiang and C. Hsiao-hwa, "Security in wireless sensor networks," *IEEE Wireless Communications Magazine*, vol. 15, pp. 60–66, Aug. 2008.

[41] K. Sun, P. Ning, and C. Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 264–277, ACM, 2006.

[42] H. Song, S. Zhu, and G. Cao, "Attack-resilient time synchronization for wireless sensor networks," *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, Nov. 2005.

Appendix A

TinyOS Implementation Details

A.1 TinyOS Abstraction Architecture

The TinyOS abstraction model is specified in [35], the hardware abstraction architecture (HAA) has three layers. The lowest layer is termed the hardware presentation layer (HPL). This layer is responsible for abstracting hardware-specific details, such as memory mapped I/O and interrupts. Generally, the HPL code does not maintain state but instead makes the hardware state available.

The next layer of abstraction in the HAA is the hardware adaptation layer (HAL). The HAL is tasked with providing a full-featured model of the hardware. Generally the HAL maintains some state, for example, to resolve sharing of the hardware when accessed from multiple higher-level software components.

The final layer of the TinyOS abstraction is the hardware interface layer (HIL). The HIL is responsible for abstracting away the difference between devices and presenting them as an instance of the same generic device model. Generally, the a HIL driver implements only a subset of the hardware capabilities exposed by the HAL in an effort to present a common-denominator interface usable across many platforms.

The TinyOS implementation structure follows the HAA strictly. One important side-effect of this is that TinyOS maps most hardware interrupts to non-preemptive task invocations.

A.2 Clock Software

As explained in [36] the most fundamental building block of the TinyOS clock software is the `Counter` interface which is used to represent a free-running counter with a given width and frequency. The `Alarm` interface extends the `Counter` interface to handle a counter with a match register that generates an event on match. Again, the `Alarm` interface has a given width and frequency.

Also outlined in [36] are a series of modules that extend the width and divide the frequency of `Counter` and `Alarm` objects. TinyOS uses these modules to create objects with the higher level `Timer` interface by transforming the lower-level `Counter` and `Alarm` objects that represent the clock hardware. Finally, the `VirtualizeTimer` module is used to allow many `Timer` interfaces to be serviced from a single `Timer` resource object by tracking the state of each virtualized timer and setting the underlying timer to expire with the next imminent virtual timer expiration.

We extended the TinyOS timer implementation by adding the high-level `Timer32khz` module to allow platform independent access to the 32 kHz clock on both the MicaZ and TelosB platforms. The underlying `HilTimer32khz` module was already present in TinyOS for the TelosB while on the MicaZ we implemented a short `HilTimer32khz` module that uses the existing `AlarmToTimer` module to transform the provided 32 kHz alarm module.

A.3 Timestamping

In addition to the addressing and data payload that are destined to be transmitted over the air, the message structures in the TinyOS network stack contain provisions to carry various metadata. The network stack manages, acts upon, and supplies these metadata, as well as supplying interfaces for the application program to interact with these metadata.

In the case of a received message, the CC2420 radio hardware must first synchronize its internal symbol clock to the transmitting radio's symbol clock. Symbol synchronization is achieved during the message preamble which consists of a series of zero symbols. Once the symbol clock is synchronized, the radio must synchronize at the byte level as each symbol represents 4 bits. Byte level synchronization is triggered upon receipt of the two-symbol long start of frame delimiter (SFD). At the end of the last symbol in the start frame delimiter, the CC2420 outputs a rising edge on the SFD pin. On the TelosB and MicaZ motes this signal is used to latch a timer value and to generate a time capture interrupt.

The time capture interrupt handler reads and stores the captured timer value. However, the captured timing data is stored in a first in, first out (FIFO) buffer separate from the network stack because at this point the packet body has not yet been received. Indeed, even the destination address has not been transmitted or received and therefore there cannot be any representation of the packet in the TinyOS network stack. Thus, the stored timestamp may be completely useless should the packet turn out to be destined for a foreign address, as the CC2420 will filter the packet at the hardware level and the packet will never appear in the TinyOS network stack.

This possibility, combined with the possibility for multiple received packets to be buffered within the CC2420 calls for some complex logic to re-unite the captured timing values with the appropriate messages once they appear in the network stack. This process includes a number of pathological ambiguous cases, such as the receipt of a packet addressed to the node in question followed immediately by two more packets only one of which is destined for the node. One of the second two packets will be dropped from the receive buffer at the hardware level but only after the preamble, SFD and address are received and a timestamp has been captured. In this case, the TinyOS software cannot determine which of the two possible stored timestamps belongs to the second locally-addressed packet. In such cases TinyOS simply stores a sentinel value in the metadata to indicate that no timestamp information is available for this packet.

In the case of packet transmission, a similar need for packet timestamping arises. Again, the CC2420 produces a rising edge on the SFD pin at the very end of the start frame delimiter byte. As in the case of reception, this rising edge causes a time capture and a timer capture interrupt. As the packet data structure is preserved until the network stack acknowledges reception to the user application, the network stack can immediately store this outgoing timestamp with the metadata for the corresponding packet.

However, the TinyOS network stack goes one step beyond simply allowing the user application to know the actual time of packet transmission. During the packet timestamp interrupt, the packet metadata can instruct the TinyOS network stack to write the packet timestamp directly into the outgoing packet data as a suffix to the usual data payload. In this case, the network stack actually changes the last few bytes of the outgoing packet in the CC2420 transmit buffer after packet transmission has started but while the CC2420 is still transmitting the packet header bytes. Obviously, if the processor is in a particularly long critical section with interrupts disabled or is busy processing higher-priority interrupts, there exists a risk that the time stamp interrupt will be delayed and the CC2420 will have already transmitted the packet prior to the outgoing timestamp being written into the transmit buffer. To allow recovery on the receiving end in this case, the packet is initially written into the buffer with a sentinel value in place of the outgoing timestamp, so that the receiver can recognize that the timestamp is invalid.

The combination of the capability to timestamp received packets and the capability to write timestamps into outgoing packets provides for direct comparison of transmitter and receiver clocks at an instant in time. This comparison is free from uncertainty caused by queueing delays in the network stack and uncertainty delays caused by clear channel assessment in the radio hardware media access control strategy.

A.4 Dallas 1-Wire Bus and Unique Identifiers

Both the MicaZ and TelosB mote design include a serial ROM programmed with a 48-bit unique identifier. The MicaZ design uses the DS2401 and the TelosB design the DS2411 unique identifier integrated circuits (ICs) from Dallas/Maxim Semiconductor. Both ICs are connected using the Dallas 1-Wire serial bus [37,38]. The Dallas 1-Wire bus uses only one bidirectional communication line from which power can be harvested for very low power devices such as the DS2401. Thus, the Dallas 1-Wire bus allows for very compact, low pin count devices.

As neither the TI MSP430 nor the Atmel ATmega128 microcontrollers have dedicated 1-Wire hardware, the 1-Wire protocol was implemented in software using a general purpose bidirectional I/O pin. The implementation followed the overall procedure suggested in [39]. The 1-Wire implementation begins at the HAL layer as there is no specific hardware state for the HPL layer to present. The HIL layer component adds block-level transfers, cyclic redundancy checking, and allows bus ownership arbitration on a first-come-first-serve basis.

The `DS24x1` module uses the 1-Wire implementation to read the unique identifier data from a DS2401 or DS2411 IC. By including the `DS24x1` module and issuing a `ReadID` request TinyOS application software can access the mote unique identifier.

The unique identifier was used in the project to allow reliable, persistent identification of the hardware between experiments. By reading the unique identifier and using it as the node network address, the project application alleviated the burden of network address management from the experimenter. Finally, by using portions of the unique identifier to seed the random number generator, different nodes start at different points in their pseudo-random sequences.

Appendix B

Security Considerations

As proposed in [4–6], the CS-MNS algorithm makes no provisions for security nor for external synchronization. In [40], the authors provide an overview of the need for, and the challenges of, security in wireless sensor networks including a discussion of the need for secure time synchronization. Some secure time synchronization schemes based on cryptographic methods have been proposed, for example in [41], where the authors use the μTESLA broadcast authentication scheme to achieve security. However, message and processing complexity are necessarily increased over unencrypted protocols.

We begin by discussing the addition of simplistic unsecured external synchronization to CS-MNS. We then proceed by analyzing the security implications of these additions and propose a two-layer external synchronization method that does not introduce large encryption overheads. The basic approach is to limit the disturbance an attacker can introduce and then to rely on algorithm robustness to mitigate the effectiveness of the attack.

B.1 Insecure External Synchronization

One simple method of adding external synchronization to a network running the CS-MNS algorithm would be to introduce reference nodes. These reference nodes would participate in the same beacon contention as CS-MNS nodes, but would broadcast the external reference clock as the beacon time stamp. These reference nodes would ignore any received beacons as they would have no need to adjust their clocks.

It is important that the reference nodes participate normally in beacon contention. In a multi hop network, increasing the probability that reference nodes win the beacon contention means that nodes neighboring a reference node have a lower beacon transmission probability. This, in turn, has the effect that nodes two hops from a reference node receive fewer beacons.

If all reference nodes are synchronized to the same external source and are functioning correctly, the time stamp beacons from these nodes will be identical and these nodes can be considered as one node for transmission. As the reference nodes discard any incoming clock samples, they can be considered as a single unreachable node for reception. Thus, whatever the true network topology, for analysis the topology can be considered to have a single reference node that can transmit to many nodes but cannot receive.

We note that these conditions are exactly the minimum connectivity requirements under which the previous analysis held; the case when there is at least one node that can communicate to all other nodes. Therefore, the addition of any number of externally synchronized reference nodes does not effect the presented analysis of behavior except to control the admissible equilibrium conditions. Preliminary simulation indicates that convergence is slowed when reference nodes are introduced into the network. However, further work is required to study the effect of differing reference node densities.

The simplicity of this external scheme may make it attractive enough for use in networks that have lax security requirements. However, requiring that the protocol allows nodes that make no clock adjustments to participate creates security concerns as explored below.

B.2 Security Through Outlier Detection

In [42], four types of attacks against network time synchronization protocols are identified: masquerade attack, replay attack, manipulation attack, and delay attack. These are in addition to more general denial of service and other jamming attacks that do not target the synchronization protocol specifically.

However, in CS-MNS, where the messages are simple time-stamped beacons that are broadcast to all nodes in range of the transmitting node, many of the attacks become equivalent. Since the nodes need not identify themselves in the beacon, masquerading as another node is identical to fabricating a beacon. Similarly, since the beacon is a simple time stamp, replaying a message is also equivalent to fabricating a beacon. Since CS-MNS does not use multi-hop routing of individual synchronization messages, there is no opportunity for an attacker to either manipulate or delay messages en route to other nodes. This leaves one viable type of attack that can be perpetrated against CS-MNS, which is the fabrication of incorrect beacons.

One possible strategy to protect against fabricated beacons is to have each node attempt to detect and ignore outliers in the received beacon stream. As long as nodes joining the network perform coarse synchronization when they join, the time processes of all cooperative nodes are expected to be similar. The outlier rejection approach is explored in [42] using both generalized extreme studentized deviate (GESD) and a much simpler threshold approach.

The case of a fabricated beacon attack is slightly different from [42], but a similar threshold approach appears promising for CS-MNS. Restricting the maximum acceptable deviation of a received beacon limits the amount of influence an attacker can have on the system. If this influence is small enough, the system will simply tolerate the attacker. Indeed, the effect of a fabricated beacon designed to fall within the threshold is similar to the effect of clock offset errors explored above.

There remains a danger that an intelligent attacker could bias the system towards ever increasing or ever decreasing clock rates. While theoretically the network remains internally synchronized, an unbounded increase or decrease in network clock rates would eventually cause numerical difficulties. Additionally, a group of attackers may attempt to exploit the threshold detection to partition the network clocks into groups separated by more than the threshold. Solutions to these problem remain to be explored.

Finally, outlier detection posses two problems for the simple reference node scheme outlined above. Eventually, circumstances will arise where the reference nodes are themselves outliers from the group and they then will become ineffective. Also, an attacker can prevent the reference nodes from bringing the rest of the network into synchrony with the external reference by acting as a reference node supplying an incorrect reference.

B.3 Securing External Synchronization

A different approach to external synchronization that is more compatible with security concerns is to allow the CS-MNS algorithm to run with simple outlier detection but otherwise unsecured. This retains the simplicity of the algorithm and avoids any encryption overhead. If reference nodes act as standard nodes and run the CS-MNS algorithm, the reference nodes can then calculate rate and offset correction factors between the internal network clock and the external clock. At this point, external synchronization of the network is a matter of disseminating this correction data to the non-reference nodes.

The presence of reference nodes in the network actually protects against coordinated attacks that aim to partition the network. As long as each partition contains a reference node, even when the network clocks between partitions do not agree, the reference nodes will distribute appropriate corrections to each partition to synchronize all nodes to the external reference.

As the CS-MNS internal network clock adjusts smoothly over time, the correction factors calculated by the reference nodes can be updated at a slower rate than the beacon rate required to synchronize the network. Therefore, more energy intensive cryptographic procedures can be tolerated to validate this information. For example, a signature scheme based on asymmetric cryptography would make it difficult for an attacker to introduce incorrect corrections even if the attacker had full access to a non-reference node.

If the corrections from each reference node are disseminated through the network, each node can defend against counterfeit reference nodes by eliminating outlier correction factors and using the average of accepted correction factors. This type of algorithm can be chosen to tolerate a given ratio of attackers to reference nodes which may be sufficient for many applications.

B.4 Preliminary Simulation Results

Figures 30 and 31 show preliminary simulation results for CS-MNS using threshold beacon filtering. The acceptance window was heuristically set at a constant $\pm 100\mu$s of the adjusted clock at the receiving node. This acceptance window is smaller than the initial offset errors and initially falsely rejects beacons from some co-operative

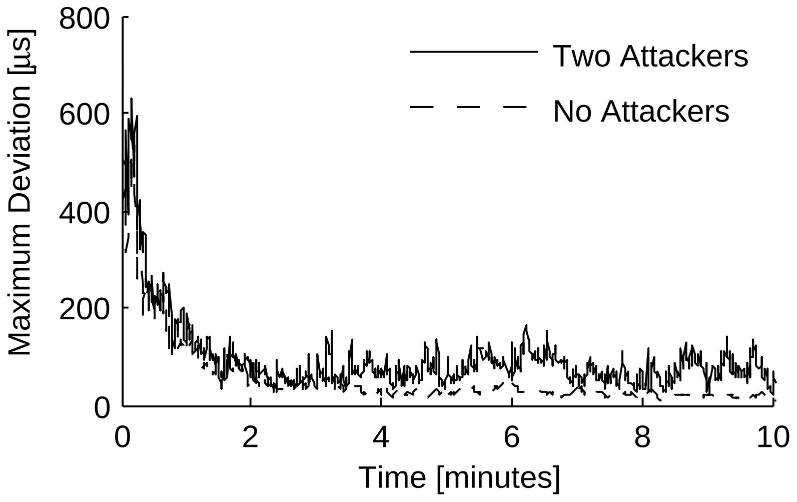


Figure 30: Simulation results for a 5 × 5 regular grid network with and without attackers located at opposite corners with random intentional errors in their beacon values.

nodes.

However, by modifying the beacon contention mechanism, adverse effects of these false rejections can be mitigated. If a node rejects a received beacon, the node continues to compete for beacon transmission instead of backing-off until the next beacon period. This allows multiple beacons to be transmitted per beacon period when the population of network clocks is widely dispersed and increases the likelihood that a node will receive a beacon within its acceptance threshold.

Figure 30 shows that an attacker that respects the protocol but introduces a random offset error into their beacon data can continually disrupt synchronization. However, the disruption is contained within the range of the acceptance window and the overall convergent behavior of the network is not effected.

Figure 31 shows a simple attack that attempts to partition the network by introducing two malicious ‘reference’ nodes. These attackers broadcast beacons normally but do not adjust their differing, diverging clocks. The attack fails to partition the network.

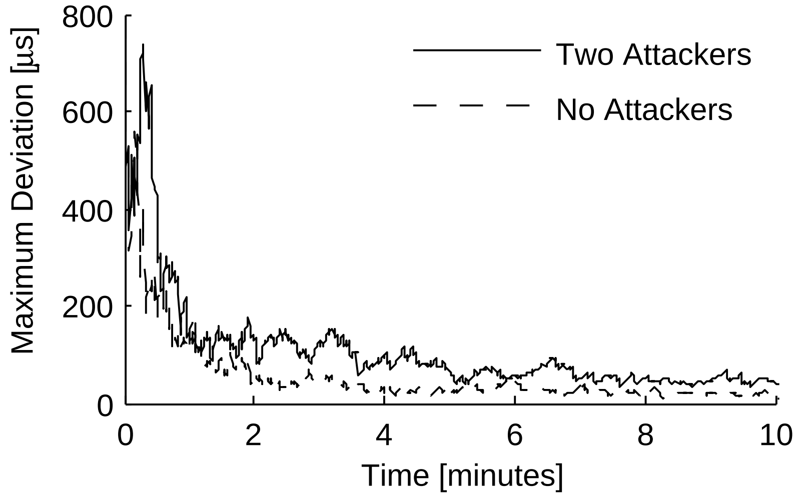


Figure 31: Simulation results for a 5 × 5 regular grid network with and without attackers located at opposite corners with differing fixed clock rates.