A Synchronous Communication System for a Software-Based Byzantine Fault Tolerant Computer

by

Reuben Marbell Sterling

B.S. Computer Science and Electrical Engineering
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Masters of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

September, 2006

Signature of Author:

Department of Electrical Engineering and Computer Science
August 21, 2006

Certified by:

Roger Racine
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by:

Barbara H. Liskov
Ford Professor of Engineering
Thesis Advisor

Accepted by:

Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

THIS PAGE INTENTIONALLY LEFT BLANK

A Synchronous Communication System for a Software-Based Byzantine Fault Tolerant Computer

by

Reuben Marbell Sterling

Submitted to the
Department of Electrical Engineering and Computer Science

August 21, 2006

in Partial Fulfillment of the Requirements for the Degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis describes the redesign of a Byzantine-resilient, quad-redundant computer to remove proprietary hardware components. The basic architecture consists of four Commercial Off-The-Shelf (COTS) processors in a completely-connected network of point-to-point ethernet connections. In particular, the focus of this thesis is an algorithm that combines clock synchronization and communications between fault containment regions by inferring relative clock skew from the arrival time of expected messages. Both a failsafe and a fault-tolerant algorithm are discussed, though the fault-tolerant algorithm is not fully analyzed. The performance of a prototype and the failsafe synchronization algorithm are discussed.

Technical Supervisor: Roger Racine
Title: Principle Member Technical Staff

Thesis Advisor: Professor Barbara H. Liskov
Title: Ford Professor of Engineering, MIT Computer Science and Artificial Intelligence Laboratory

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgment

August 21, 2006

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Internal Company Sponsored Research Project 20317-001, Software Based Fault Tolerant Computer.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions herein. It is published for the exchange and stimulation of ideas.

_____

(Reuben Marbell Sterling)

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

Chapter 1

Introduction

The advantage of fault tolerant computer systems in critical applications is clear. Computers are a powerful and ubiquitous resource, and their reliability can dictate the success or failure of a system. In particular, applications involving human safety require computer reliability far greater than what is typically needed. In such important applications, fault tolerant computer systems are absolutely necessary to ensure the safety of the humans involved.

In a naively engineered system, reliability of the whole system is only as good as the most unreliable piece. Unfortunately, building basic components of a system to the standards required by critical applications is usually infeasible due to time, cost, or technological constraints. As a result, systems must be engineered with the low reliability of individual components in mind.

The reliability of a computer system depends on both the correctness of the software written for the system and the robustness of the hardware against malfunction, particularly when the system operates in extreme environments like Earth orbit and beyond. Both hardware and software have their own unique challenges when implementing highly-reliable systems. This thesis is concerned primarily with techniques to increase the reliability of

the hardware aspects of critical systems by running identical code on redundant computers.

1.1 Thesis Description

This thesis project spans the first year of a two year effort to redesign an existing fault tolerant computer to remove specialized hardware. The original fault tolerant computer, built by Draper Laboratory for NASA's X-38 vehicle, features four redundant computers that communicate via special-purpose hardware. The system also provides a complete API for user applications to take advantage of the fault tolerance features. The result of the current thesis work is a prototype fault tolerant computer running entirely on generic hardware, with the specialized communication protocols implemented entirely in software. Basic abstraction layers were built to support a rudimentary API, and an experimental application was developed to demonstrate the control of a simple inverted pendulum.

The work focused on redesigning the low-level synchronization, communication and voting protocols that support the fault tolerant properties of the system, since this is the functionality implemented at the hardware level in the X-38 computer. While the algorithm used to vote the inputs and outputs of the redundant computers remains largely identical to that implemented in the X-38 system, the synchronization and communication protocols are substantially different.

In addition to the implementation of a prototype system, a major goal of this thesis was to propose and develop a Byzantine resilient synchronization algorithm targeted at the specific requirements for this system. The algorithm runs on each node and determines how far to adjust that node's clock forward by inferring other nodes' clocks from the arrival times of messages. At each round of execution, the algorithm selects the node that it determines to be the most ahead and adjusts the local clock to match it as closely as possible. Unfortunately, due to time constraints, the proposed algorithm was not developed

sufficiently. Instead, the partial work completed toward the development of the Byzantine resilient algorithm is given in this thesis, and the continued development and comparison is left for future work.

1.2 Advantages of Software-Based Fault Tolerant Computers

Traditionally, fault tolerant systems are built using specially designed hardware. This specialized hardware might act as interfaces between redundant computers, or it may even be processors designed for fault tolerance at the CPU level via redundant circuitry. Hardware-level fault tolerance is motivated by performance requirements typically not achievable at a software level, but the development and low-volume fabrication costs of special hardware makes fault tolerance typically quite expensive. Fortunately, as processor speeds grow and communication latencies shrink, the opportunity to lift fault tolerance functionality from hardware to software becomes more realistic. The migration of fault tolerance support to software allows the use of generic hardware. As a result, fault tolerant computers will become cheaper to produce.

In addition, implementing fault tolerance support in software allows an unprecedented level of flexibility. For instance, if a hardware-based fault tolerant computer loses some specialized hardware due to a fault, the functionality of that piece of hardware may be lost for good. A fault tolerant computer in which specialized functions are implemented in software could migrate that functionality from one generic piece of hardware to another.

1.3 Organization

Chapter 2 presents a discussion of background information on the subject of fault tolerant computers and clock synchronization. Chapter 3 follows with a description of the design of the prototype software-based fault tolerant computer (SBFTC) developed as part of this thesis work.

Chapter 4 describes the partially-developed Byzantine resilient synchronization algorithm. It begins by describing the motivation for the development of the algorithm and then introduces the definitions and assumptions used in the design and analysis. A simpler, failstop (non-Byzantine) version of the algorithm is presented and analyzed as a introduction and guide to reasoning about the Byzantine resilient algorithm. Finally, the Byzantine resilient algorithm is introduced. The intuition behind the algorithm is described, and the beginnings of mathematical analysis are presented, but a full analysis is not yet available.

Chapter 5 describes the experimental application developed to demonstrate and test the prototype SBFTC, and, finally, Chapter 6 presents the experimental results from this thesis work and concludes with suggestions for future development.

Chapter 2

Background

This chapter provides background information necessary to understand the thesis topic before the work is described in greater detail. An introduction to fault tolerance, particularly Byzantine fault tolerance, is presented, followed by an introduction to the Draper Laboratory X-38 fault tolerant computer. A brief survey of former research in clock synchronization is given, along with additional relevant research.

2.1 Basic Voting Architectures

Techniques to increase the reliability of the hardware of critical systems typically use redundant computers, each executing the same code simultaneously, operating on the same inputs, and (ideally) producing the same outputs.

Figure 2-1 presents a logical diagram of such a system. Fault detection is done via voting. When the redundant computers produce output, such as an actuator command or status message, the outputs from all the redundant computers are compared by a voting element for discrepancies. If a discrepancy is detected, various actions may be taken, depending on the number of redundant outputs available for comparison. Assuming only a single er-

ror, if three or more redundant outputs are available and two are in agreement, the system may continue to function normally. If only two redundant outputs are available and they disagree, typically the system cannot continue and merely reports an error and ceases execution. This system behavior stands in contrast to a non-redundant system, which would continue unchecke[illegible]rror.

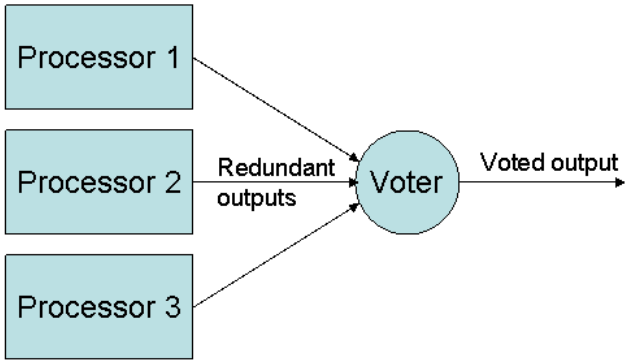

Figure 2-1: In the basic voter architecture, multiple computers execute the same code simultaneously. The redundant output created by the computers is voted to produce a single output masking any single error.

2.2 Advanced Voting Architectures

The voting system described above is not sufficient for many critical applications, particularly space operations, where radiation can corrupt any part of the system. Although faults that occur within the redundant computers would be handled correctly, consider the consequences of a failure of the voter element. While the voter may be considerably less complex than the redundant processors and less prone to error, the existence of a single point of failure excludes this design as a viable candidate for space applications.

Figure 2-2 shows an architecture that avoids single points of failure. Each node in the redundant system represents a fault containment region (FCR). Each FCR experiences

faults independently from every other. For example, an electrical short in one FCR is guaranteed not to affect any other FCR.

Every node is fully connected to every other via a bi-directional link. Messages can travel from one node to another without affecting communication in the opposite direction or between any other pair of nodes. While single failures may cause an FCR or a link to become unavailable, this architecture supports communication and voting protocols that are unaffected

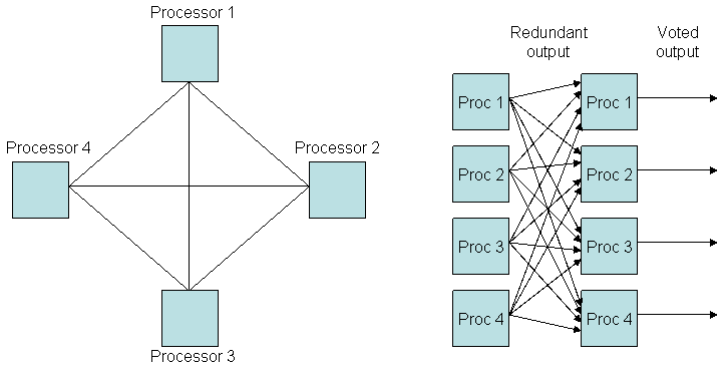

Figure 2-2: In the advanced voter architecture, processors broadcast their output to all other processors, which each vote the results.

2.3 Introduction to Byzantine Resilience

One important type of fault tolerance supported by the above architecture is known as Byzantine resilience. The term “Byzantine” is used as a synonym for “arbitrary.” It was originally coined in a paper by Lamport, et. al. [3], which described algorithms to reach consensus among cooperating parties – in their examples, Byzantine military generals – despite the existence of “traitors” that may fail to pass on messages, lie, or even conspire to break the consensus among the loyal generals.

The ability to tolerate arbitrary system faults, including lying and collaboration, might

seem like an excessive precaution. In many kinds of applications for which fault tolerant systems are required, like manned space exploration, the likelihood that the system may contain an active adversary is slight. Nonetheless, there are a few reasons why it might be preferable to design a system to be Byzantine resilient.

First, many types of failures that can occur naturally are similar to the types of attacks an active adversary might use to break the consistency of a system. For example, a node with failing hardware could easily transmit slightly different versions of a message to different recipients, introducing the same sort of confusion to the system an active adversary would want. Second, designing a system to handle *any* fault makes reasoning about the capabilities of the system much easier.

Lamport's paper proposes the following scenario. In a group of n generals, there exists a single commanding general and $n - 1$ lieutenant generals. The commanding general must send a message to his lieutenants such that the following requirement are satisfied:

1. All loyal generals obey the same order.
2. If the commander is loyal, every loyal lieutenant obeys the order he sends.
3. If the commander is not loyal (sends conflicting messages), every loyal lieutenant must agree on the same order (or agree to take no action).

In the architecture shown in figure 2-2, the incorrect execution of any single processor is considered a single fault. When a system is described as Byzantine resilient, it means that the system can tolerate some number of arbitrary faults and still function correctly. The number of tolerated failures depends on the number of nodes in the system, how completely the nodes are connected, whether the system is synchronous or asynchronous, and also on the types of messages being passed.

A synchronous system is one where the absence of a message from a particular node implies that the node is faulty. An asynchronous system makes no such assumption. The

Internet is an example of an asynchronous system. If a computer sends a message to another computer across the Internet, the communication medium offers no guarantees about the delivery time of that message or even that the message will arrive at all. A failed delivery does not implies that the sender is faulty. Systems with more predictable communication mediums, such as point-to-point connections, can support synchronous communication. If a node expects a message to arrive from another node by a particular time and it receives no message, the receiving node may assume the sender is faulty. Lamport et. al. assume synchronous communication.

Lamport et. al. describe two message types, “oral” and “written”. Intuitively, oral messages may be modified as they pass through nodes. Written messages, however, cannot be changed without detection and are always “signed” by the author. As a result, a recipient of a written message can know for certain who composed the message, or else he is guaranteed to notice a modification.

For synchronous systems using oral messages, Lamport et. al. show that in order to tolerate f faults, there must be at least $3f + 1$ total nodes in the system, i.e. $2f + 1$ loyal nodes. The system described in this thesis implements oral messages and a synchronous communication system.

A Simple Byzantine Resilient Protocol

Consider a system designed to handle f faults. There must be $n \geq 3f + 1$ nodes in the system. Each pair of nodes has a dedicated two-way link through which that pair can send messages between each other. Thus configuration allows a node to distinguish between messages from different nodes. The generalized algorithm requires f recursive iterations, but the specific case of $f = 1$, which only requires two communication rounds, is described here.

The steps of the algorithm are as follows:

1. The node with the message to transmit begins by broadcasting the message to all other nodes.

2. When a node receives the original message, the node stores it and re-broadcasts it to all other nodes.

3. When a node receives the re-broadcasted message, the node stores it.

4. After both communication rounds are complete, each node compares all the versions of the messages it received and accepts the majority version. If no majority is found, no message is accepted.

This algorithm satisfies the requirements listed above for Byzantine resilience. Consider requirement 2. Given that only one adversary may exist, if a commander sends the same message to all lieutenants, every node will receive the same message *at least* one more time. Since each lieutenant receives a maximum of three versions (from the commander and two other lieutenants), two matching messages comprise a majority, and the correct message is accepted.

Now consider requirement 3. A non-loyal commander is one who does not send the same original message to all three lieutenants. Fortunately, since the commander is dishonest, the three lieutenants are guaranteed to be honest. Let m_1, m_2 and m_3 be three possible messages the commander can send. Say the commander sends m_1 to two of the lieutenants and m_2 to the third. Then, in the second round, the first two lieutenants will receive another copy of m_1 from the other and the third will receive *two* copies of m_1 from the first two. Thus, all three lieutenants have received a majority of m_1 messages.

Suppose the disloyal commander sends m_1 to the first lieutenant, m_2 to the second, and m_3 to the third. This time, after round 2, none of the lieutenants will have received a

majority of *any* message, thus all the lieutenants will agree to not accept any message.

Requirement 1 follows from 2 and 3.

2.4 Introduction to X-38 Fault Tolerant Computer

The X-38 fault tolerant computer (FTC) is a Byzantine resilient, quad-redundant system on which this thesis is based [8]. Implemented at Charles Stark Draper Laboratory in 2002 for NASA's X-38 experimental aircraft, it provides a flexible architecture in which fault tolerant properties are provided by specialized hardware elements known as Network Elements (NE). In addition to the fault tolerant hardware architecture, the X-38 computer includes a large set of software libraries, which high-level software applications use to inherit the fault tolerant properties of the system. Since this thesis is primarily concerned with redesigning the specialized hardware as software, only the X-38's hardware architecture is described here.

Five NEs are linked in a fully connected network, and they perform the synchronization and I/O voting necessary for fault detection. The actual redundant processing is done by COTS processors that sit behind the NEs. Each redundant processor resides behind a different NE. A group of redundant processors is known as a virtual group. The quad-redundant virtual group requires processors residing behind four NEs, but the fifth NE still participates in communication/voting, allowing the system to tolerate two non-simultaneous faults. The first fault may be a Byzantine fault. After the system has recognized and recovered from the first fault, it is prepared to handle a second. Under certain conditions, the second fault may also be Byzantine, but under others, the system may only handle a subset of possible second faults.

The flexibility in the design is reflected by the ability to add an arbitrary number of COTS processors behind each NE, with the processors and the associated NE connected

through a VME backplane. As a result, multiple redundant computers can all exist simultaneously in the X-38 architecture, with each NE handling the communication for processors belonging to one or more redundant computers. A diagram of a potential hardware layout is given in figure 2

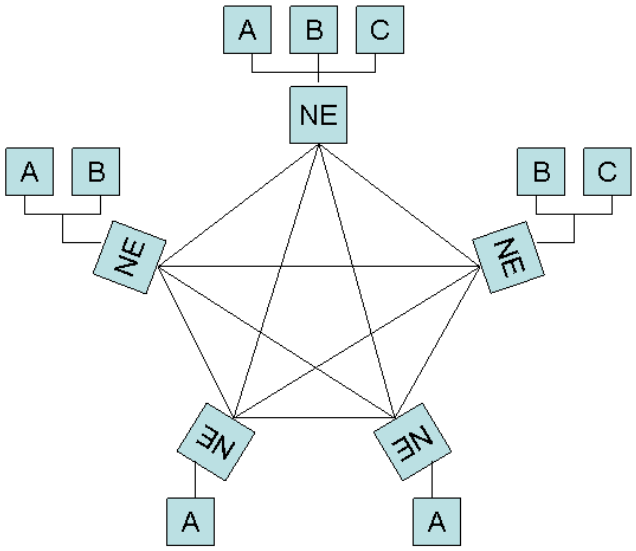

Figure 2-3: The X-38 is capable of supporting several redundant computers. Each processor of a redundant computer resides behind a different NE. In this example, processors marked by A are part of a quad-redundant computer, processors marked by B are part of a tri-redundant computer, and processors marked by C are part of a duplex computer. Simplex computers (not shown) are also supported.

The actual X-38 hardware layout contains only one quad-redundant computer and five simplex (non-redundant) computers. A diagram depicting the actual hardware layout of the X-38 computer is given in figure 2-4. The simplexes perform the roles of actuator and sensor control and communicate with the quad-redundant computer through the fault tolerant communication services provided by the NEs. Each quad-redundant processor was referred to as a flight-critical processor (FCP), and each simplex was known as an I/O Control Processor (ICP).

Communication between the NEs occurs at regular intervals and follows what is called a

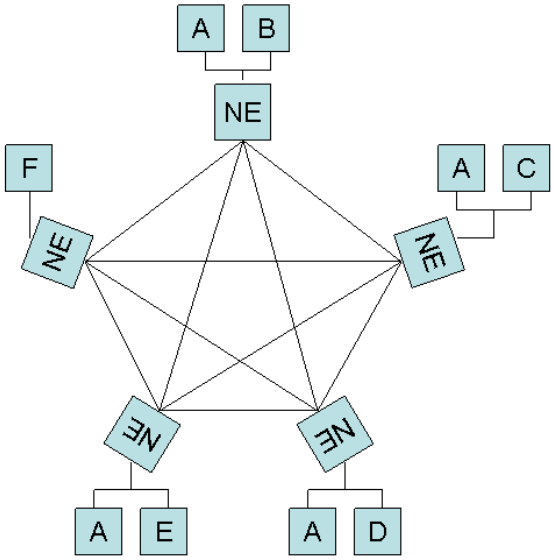

Figure 2-4: The actual X-38 configuration consists of a single quad-redundant computer (A) and five simplex computers (B, C, D, E, and F), which control actuators and sensors.

System Exchange Request Protocol (SERP). The basic feature of the SERP is an agreement step that takes place before each message exchange. During the agreement step, each pair of boards determines whether there is a message to send, whether the receiving board has space in its buffers to accept the message, and what class of message is to be sent. When an agreement is reached, the actual message is sent between the NEs. Once the appropriate voting steps among the NEs have been completed, the NEs deliver the message to the appropriate processor that resides behind them.

Since the NEs must know when to perform the SERP, a clock must be shared between NEs so they may communicate with the other NEs at the correct times. If no clock were shared, the various clocks used across the system would drift apart, eventually making the communication system useless. Unfortunately, NEs cannot directly share clock signals, since this would violate the integrity of the fault containment regions. Instead, clock synchronization is established through messages passed over the fault tolerant network.

2.5 Clock Synchronization

Much prior research has been published on the topic of fault-tolerant clock synchronization in distributed systems. One example of a distributed clock synchronization system is the Network Time Protocol (NTP),which is heavily in use throughout the modern Internet [7]. NTP is an extension of the distribute time service originally proposed by [6]. NTP is designed for a highly unpredictable environment, i.e. the Internet, where communication between computers can be delayed large and unpredictable lengths of time. NTP uses a large number of sources to determine the correct time, potentially down to milliseconds. The algorithm first determines a range of possible correct times based on a sampling of readings from a single source, with each sample taking a variable length trip across the network. Based on the intersection of a number of such ranges, the algorithm produces a good approximation of the true time. While NTP is very effective for Internet synchronization, the amount of overhead required makes it less appealing for embedded applications, where network latency can be much more predictable. Also, NTP is a "one-direction" algorithm where there is eventually a single, ultimate source of time, such as an atomic clock.

Algorithms for distributed, embedded systems commonly do not enjoy a connection to a wide area network, and thus cannot update their time from oracle sources. Instead, the nodes must cooperatively keep their time synchronized. Fortunately, such applications have much more predictable networks, which allows clocks to be kept quite synchronized, and many fault-tolerant algorithms have been developed for this purpose. The properties of these algorithms guarantee that if enough nodes are operating correctly, that all correct nodes in the system will adjust their clocks toward a consistent point in time and maintain synchronized clocks.

A general solution to the problem is given by [2]. This algorithm does not assume a completely connected network, but instead assumes that all correct nodes are fully con-

nected through some non-faulty path. The latency between any two correct nodes along the fault-free path is assumed to be bounded by some constant. The solution also requires the use of signatures. Like the algorithm presented in this thesis, neighboring clock times are inferred from the arrival times of messages, but since this solution does not assume a completely connected network, the achievable bound is not as favorable.

Another, more related solution is presented in [5]. This algorithm is very similar to the solution proposed here. Messages are sent from each node at a standard time, as measured by each node's logical clock, and every node waits long enough to receive the messages from every other correct node and the arrival time is recorded. Afterwards, an averaging function is applied to all the times at which the messages were received and the local clock is adjusted to the calculated average. The achievable synchronization is a function of the variation in network latency, i.e. minimum and maximum latencies. However, this algorithm potentially sets some clocks backwards which is not acceptable for the SBFTC.

The above algorithms strive to be optimal in terms of clock agreement between nodes. The algorithm in [9] provides synchronized clocks that are optimal in reference to accuracy, i.e. the departure from real time. The drift of the synchronized clocks from real time in this algorithm are only as much as the underlying physical clocks' maximum drift rate. However, the actual synchronization between nodes is not optimal. The maximum synchronization is a function of the maximum network latency, not of the variation in network latency. This algorithm follows the requirement that clocks may never be adjusted backwards.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

System Overview

The aim of this thesis work is to redesign the previously existing X-38 fault tolerant computer architecture to remove proprietary hardware components. Since the X-38 proprietary hardware's role is fault-tolerant communication between fault containment regions (FCR), i.e. a Network Element (NE) and its associated processors, the principal contribution of this thesis is to implement the fault-tolerant communication protocols as software running on COTS hardware. This style of computer can be referred to as a software-based fault tolerant computer (SBFTC).

An SBFTC offers some advantages over an FTC containing proprietary hardware. One clear advantage is cost. Producing a system requiring proprietary hardware is a costly endeavor, requiring greater development resources and low-volume fabrication of the specialized hardware. Of course, the high development and manufacturing costs imply a less competitive price on the market. A software-based solution promises faster development and lower-cost hardware, since it can be purchased from a third-party vendor, who presumably manufactures the product in higher quantities and can sell it cheaply.

Another advantage is flexibility. As more of a system's functionality is raised to the software level, the less dependent on any one piece of hardware the system becomes. If a

node fails due to a physical malfunction, a software process might be moved from executing on the failed hardware to another spare.

Traditionally, SBFTC's have been infeasible due to technology constraints, particularly CPU speed. Even with the advent of modern high-speed processors, radiation-hardened technology required for space operation tends to lag in performance. For example, while current high-end processor technology approaches clock speeds of several gigahertz, radiation-hardened computer clock speed remains in the range of hundreds of megahertz.

The SBFTC developed as part of the thesis work is a prototype intended to test the feasibility of a such a system with current technology. Since the SBFTC is intended to replace the original X-38 FTC, the design strives to achieve the specifications of the original system: two non-simultaneous faults, arbitrary numbers of virtual groups, and similar computation power and I/O throughput.

The system currently only handles a single fault, since it has only four nodes and implements only oral messages. For work on how to augment the SBFTC with written messages to support two faults, see [1].

3.1 Physical Description

This section describes two architectures. The first is the ideal fault-tolerant architecture toward which the work described in this thesis is targeted. The second is the actual physical architecture used in the experimental application of this thesis work. As will be seen, the actual physical architecture does not provide the properties of the ideal architecture and, therefore, is not well-suited for the application. Fortunately, the software written for the application hides the ill-suited architecture via an abstraction layer, so the high-level software described in this section and the synchronization algorithm described in Chapter 4 are written as if running on the ideal architecture. Unfortunately, the performance of the

system is adversely affected.

The physical layout of the ideal SBFTC consists of five boards, each with its own CPU. Four of the boards run both an NE and a flight-critical processor (FCP). The remaining board runs only an I/O Control Processor (ICP). The four NE/FCPs have independent, point-to-point connections. Messages sent on one link will not cause contention with messages on any other. The fifth ICP board only shares a connection with a single NE. If the ICP wishes to send a message to any other board, it must send the message first to its associated NE, which will forward it to the other nodes. Figure 3-1 depicts the ideal architecture of the SBF

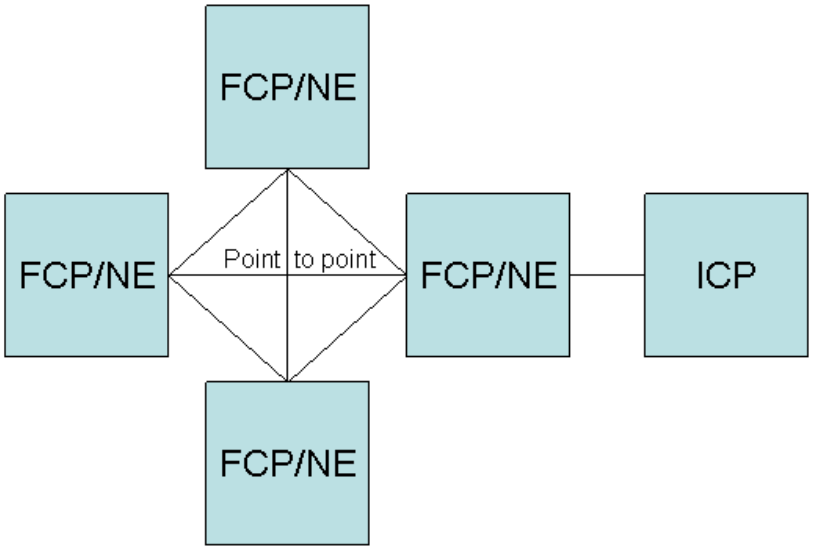

Figure 3-1: The ideal SBFTC would be composed of four nodes connected over a point-to-point communication medium. The fifth node, an ICP, may only communicate with its associated NE.

The actual physical layout of the SBFTC also consists of five boards, an ICP and four NE/FCPs, but all five boards are connected via a common ethernet. Each board, an Embedded Planet EP405, features a 300 MHz IBM PowerPC 4xx processor and a 100BaseTX ethernet adapter. Figure 3-2 depicts the actual physical layout of the SBFTC.

There are a few noteworthy departures from the ideal design. First, there is a common

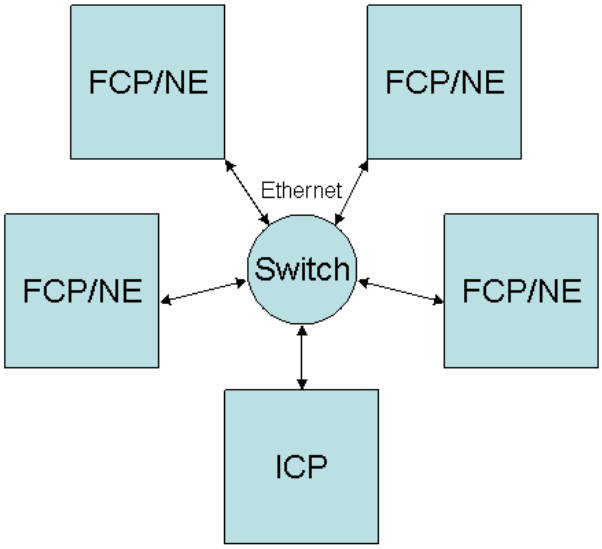

Figure 3-2: The actual SBFTC is composed of five boards with ethernet adapters and one ethernet switch, which provides a common network between all the boards.

ethernet switch connecting all the boards instead of a dedicated channel between each pair of boards. Clearly, a common switch for all inter-board communication introduces a single point-of-failure and should not exist in an ideal system.

Additionally, ethernet network contention raises the issue of increased, and even indefinite, network delays. The switch alleviates some of the problem. Typically, every board connected to an ethernet competes to send its message across a shared network. If two boards try to send a message at the same time, the boards detect the conflict, and each board waits a random amount of time before trying again. With so many nodes trying to send messages at the same time, the potential for large delays is high. In fact, theoretically, an ethernet does not guarantee that a message will ever reach its destination, since there is an exponentially small probability that nodes may continue to attempt sending at the same time forever. The switch solves this problem by accepting messages from multiple nodes simultaneously and storing the competing messages in a buffer, thus preventing the boards from ever conflicting and waiting. However, messages with the same destination

still need to be sent serially across the same wire. As a result, messages sent at the same time may take a long time to reach their destination. Fortunately, if there is a known upper bound on the number of messages in-flight at any time, there is a known upper bound on the maximum latency. Figure 3-3 illustrates how the switch handles messages.

Due to resource constraints, the common switch is the easiest way to simulate a completely connected point-to-point network. Fortunately, as long as the network can guarantee a maximum bound on latency, there is no theoretical limitation to the system capability other than decreas

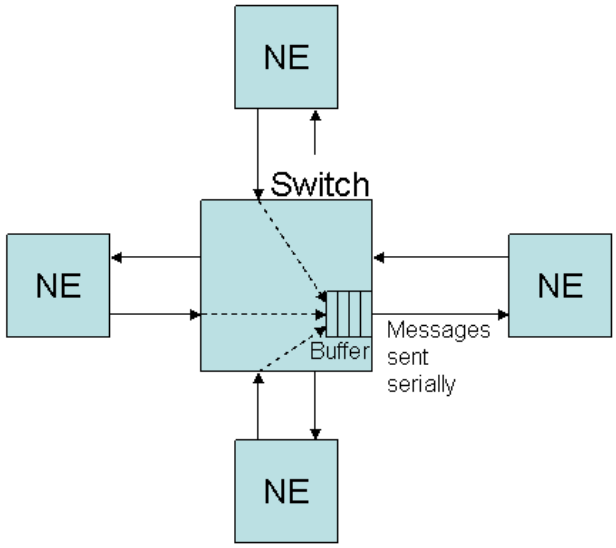

Figure 3-3: The common switch allows all boards to send at the same time and buffers the messages internally. Messages with a common destination must still be sent serially after they are buffered.

Another curious feature is that not all the nodes connected to the common network are NEs. Four of the nodes are NEs and the last is an ICP. Recall that in the X-38 FTC, ICPs and FCPs both reside behind NEs. In the SBFTC, this single ICP should also exist behind an NE and not have direct access to the common network, but, again, lack of access to the required hardware necessitated the given topology. Logically, the ICP functions as if it were located behind a single NE via software-layer abstraction.. It does not participate in any

communication protocols between NEs and only communicates with a single, designated NE, but it uses the common network to do so.

3.2 Software Design

Figure 3-4 depicts the hardware and software components of a single FCR of the X-38 FTC and the new SBFTC and demonstrates a few major differences between the two designs. First, and most importantly, the network element now exists as a software module running on a commercial off-the-shelf (COTS) board. Also, while the NE hardware in the X-38 FTC is dedicated entirely to performing NE functions, the COTS board in the SBFTC runs both t

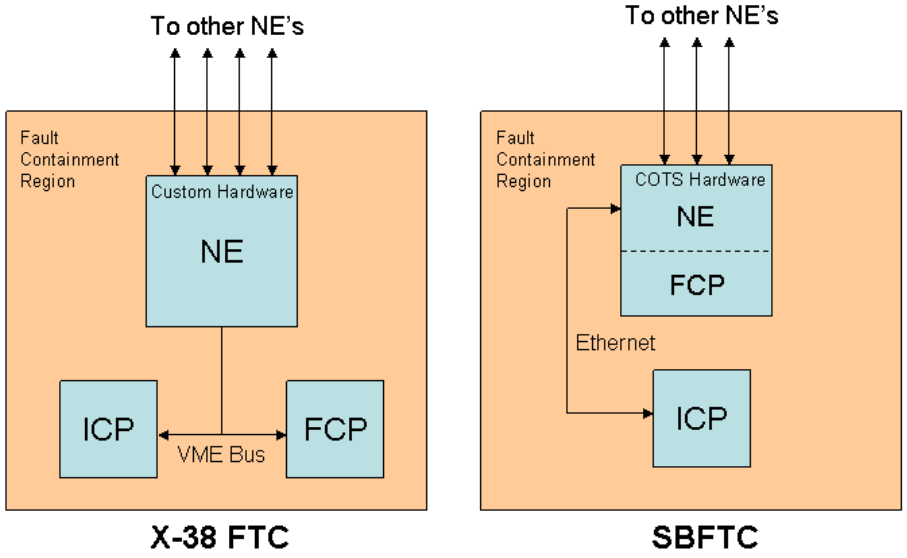

Figure 3-4: The physical layout of a single FCR in the X-38 FTC and the SBFTC. In a shift from the X-38 FTC design, the SBFTC combines the NE and the FCP as logical processors executing on a single board. The ICP still runs on a separate board.

Both FTCs use a real-time operating system (RTOS) to help ensure that critical tasks receive sufficient CPU resources at the appropriate times. The RTOS used for this project

is Green Hills Integrity. Integrity is marketed as a “time and space partitioned” RTOS. Its time partitioning helps support “hard” real-time constraints. A system with tasks that *must* be completed by given deadlines (or else the system fails) is said to have hard real-time constraints. In contrast, a system with more relaxed deadlines is said to have soft real-time constraints. Integrity’s time partitioning guarantees that software modules receive the amount of CPU time intended by the system designer regardless of the behavior of other modules running on the system. Its space partitioning guarantees that a software module cannot interact (or interfere) with other software modules running on the same board unless given explicit permission before runtime. As a result, despite accidental or malicious behavior of foreign software, the kernel and other software processes remain unaffected.

A time and space partitioned RTOS is particularly useful in the software design of the SBFTC, since it allows the NE to reliably coexist on-board with the FCP. Not only must the NE be protected from the FCP software, which contains application-dependent, foreign code, both the NE and the FCP must perform time-critical operations despite sharing the CPU with one another. A time-partitioned RTOS helps guarantee that the CPU is shared correctly and efficiently.

3.2.1 Abstraction Layers

Abstraction layers in the SBFTC hide low-level implementation details of the system from higher-level software components. This allows cleaner, clearer code, and it also will allow the non-ideal physical layout of the system to be changed in the future without modifying the higher-level code. The SBFTC implementation defines a few layers of abstraction, depicted by figure 3-5.

The lowest layer of abstraction in the SBFTC is the board layer. A board is the basic

unit of hardware, and there are always a fixed number of boards in the system. Code written at this layer is aware of the physical layout of the system.

The virtual processor (VP) layer sits above the board layer. The VP layer defines three processor types: NEs, FCPs and ICPs. One or more VPs may exist on a single board. The VP layer keeps track of which processors are running on which physical elements. In the current prototype, there are a fixed number of NEs, FCPs and ICPs in the system, and the particular boards where they exist are set before run-time and remain static throughout the lifetime of the system.

The virtual group (VG) layer is built on top of the processor layer. A VG is a set of one or more VPs, each sitting behind a different NE, that act as a single, logical computer. Each member of a VG executes the exact same software, operates on exactly the same inputs, and produces, under error free conditions, exactly the same output. Several different VG's may exist in the SBFTC at the same time, and different VG's may contain different numbers of members. This software layer maintains knowledge of which processors comprise which groups. In the prototype, VGs and their members are set before run-time and remain static throughout the lifetime of the system. Ideally, the system should be capable of assembling VGs upon startup according to available resources and reconfiguring as resources change during execution. Such an extension is left for future development.

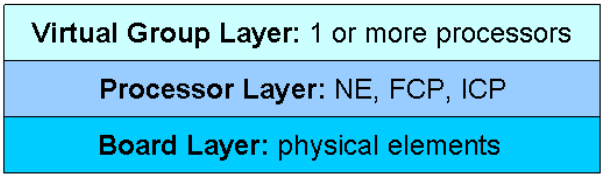

Figure 3-5: The SBFTC software implements several abstraction layers, the hierarchy of which is depicted here. The board layer recognizes individual, fixed physical elements is the system. The virtual processor layer recognizes processors running on top of the physical elements. The virtual group layer combines multiple processors into redundant, fault tolerant computers.

3.2.2 Communication Stack

Each abstraction layer described in Section 3.2 has an associated communication layer. The communication stack defined for the SBFTC sits atop the layers defined by the Open Source Interconnect (OSI) model (physical, link, network, and transport). Each communication layer provides an API to the layer above, which uses the API to route messages to the corresponding layer on a destination node. The following sections describe the relationships b[illegible]re 3-6.

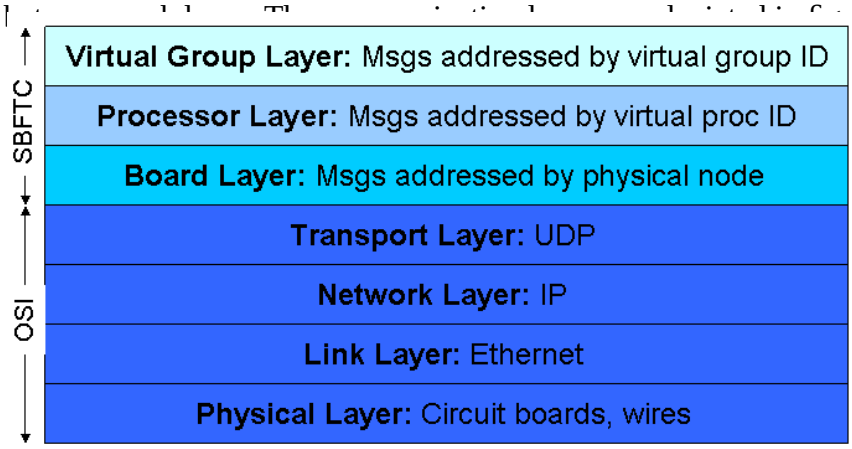

Figure 3-6: Each SBFTC abstraction layers defines an associated communication medium, the hierarchy of which is depicted here. The SBFTC layers are built above the Transport OSI layer. Messages between virtual groups are addressed by virtual group ID. Virtual group messages are turned into processor layer messages and passed to the processor layer. The processor layer maintains a list of which virtual processors are running on which physical nodes, and passes appropriately addressed packets to the board layer. Each layer may require some message processing at each hop to determine if the destination has been reached yet.

Board Communication

Boards are the lowest abstraction layer defined by the SBFTC. Messages between boards are passed via UDP packets, which are sent over a switched 100 Mbps ethernet network. Each board contains a standard ethernet adapter with a unique ethernet address and unique

IP address.

Ideally, each NE would have a dedicated point-to-point connection to every other node. Such a design is simulated by connecting each NE through a shared ethernet switch. The switch eliminates wire contention between the ethernet adapters at the multiple nodes, thereby reducing the average transport latency across the network.

In an ideal system, dedicated point-to-point connections would be achieved with multiple communication ports on each board (not necessarily ethernet), with each port connected directly to that of another board. If this were the case, it would be the responsibility of the board communication layer to route messages destined for a particular board through the correct communication port.

In the interests of prototype performance, it should be noted that circumventing the transport and network layers entirely and using the ethernet link layer directly may be desirable, however such optimizations are beyond the scope of the thesis.

Virtual Processor Communication

NEs, FCPs and ICPs pass messages via a communication layer that sits above the board communication layer. If the board communication layer is analogous to the OSI model's physical layer, the VP communication layer is analogous to the OSI link layer. It provides a protocol to send messages across the board layer to an "adjacent" VP.

FCPs and ICPs are considered to be adjacent to their associated NEs, and NEs are adjacent to each other. This communication layer cannot be used to pass messages between FCPs and ICPs. Such communication is required to be fault tolerant, and it is handled via virtual groups at the next communication layer.

Each NE is assigned an ID that is unique to the system, and each FCP or ICP is assigned an ID that is unique to its associated NE. The IDs are assigned before run-time and remain constant throughout the lifetime of the system. The processor layer on each NE stores a

table mapping VP IDs to board IDs. When a message is sent using this layer, the board ID is fetched from the table, and the message is passed down the communication stack to the board layer, which sends the message over UDP to the correct board.

A message originating from an FCP or an ICP need not include an address since it may only send messages to its NE. A message originating from an NE may be addressed to one or more NEs or to a single VP. NEs may address messages to themselves.

In the current prototype, messages must be a fixed length, although this limitation was primarily imposed for ease of implementation and analysis. Although there is no fundamental minimum limit to the length, the nature of the communication network requires that a maximum message length be imposed for timing reasons.

Since multiple messages may need to be sent from one NE to another during the same interval, part of the NE/NE communication routine's job is to merge these messages into a single buffer when transmitting and to split the messages when receiving.

NE/NE messages are exchanged regularly at fixed intervals, and all NEs send their messages at as close to the same time as possible. Small differences in send times are due to discrepancies in each board's internal clock and fundamental limits of synchronization.

The strategy of sending all messages at the same time is advantageous in a few ways. First, it eases analysis of the system and allows an easy bound to be determined for the latency of messages across the network. Also, it provides the opportunity for clock synchronization based on inferring clock differences between boards via message arrival times. The clock synchronization algorithm used in the SBFTC is presented in Section 4.

FCP/ICP to NE messages can be sent asynchronously. NE to FCP/ICP messages are sent by the NE at regular intervals following NE/NE message exchanges in order to deliver incoming messages.

Virtual Group Communication

The virtual group (VG) communication layer is analogous to the OSI network layer, which sits above the link layer and provides communication between non-adjacent nodes. The VG communication layer provides communication between non-adjacent VPs.

Each VG is associated with a system-wide unique ID. In the prototype SBFTC, VG's and their members are hard-coded into the system. Ideally, the system should be capable of assembling VG's upon startup and reconfiguring during runtime according to available resources. Such an extension is left for future development.

VGs provide the basic level of Byzantine fault-tolerance in the SBFTC. As discussed in Section 3.2, a VG is one or more VPs all running the exact same code simultaneously, operating on the same inputs and producing the same outputs. This communication layer enforces that communication occur between entire VGs instead of between single VPs. Any message sent from one VG to another will be delivered reliably, provided that the system currently is suffering at most one fault.

The guarantee of reliable delivery requires:

1. The output of all members of a VG are correctly voted to mask errors in any one of the members of the VG.

2. The same input is delivered to all members of the recipient VG.

To achieve these requirements, the SBFTC uses a broadcast and reflect algorithm similar to that discussed in Section 2.3. There are two kinds of messages that are sent over the VG communication layer, known as class 1 and class 2 messages. Class 1 messages are also known as single-source messages and are used when a single-member VG (an ICP) sends a message to a multi-member VG (an FCP). An example of such a message is an input sensor sending data to a set of FCPs. Class 2 messages are sent when a multi-member

VG sends a message to another VG of any size, though typically an ICP. An example of such a message is a set of redundant FCPs belonging to the same VG sending a control command to an ICP controlling a flap or an engine valve. ICPs may not send messages to other ICPs, so no message class is defined to handle such cases.

Class 1 Messages

Figure 3-7 shows the path of a class 1 message. A class 1 message makes four hops in its journey from an ICP to multiple FCPs.

1. The message travels from the ICP to its associated NE.
2. The NE recognizes that the message originated from an ICP, initiates a class 1 message exchange by sending a copy of the message to all other NEs in the system.
3. Each receiving NE responds by reflecting its copy of the class 1 message to all other NEs.
4. Each NE determines the correct message by comparing its multiple copies. Each NE then checks whether it is responsible for a member of the destination VG indicated is the message. If so, the NE forwards the message to the appropriate FCP.

At the end of the exchange, each FCP receives the exact same message at approximately the same time.

Note that all NEs vote their multiple message copies regardless of whether they are responsible for an FCP on not. This way, in the event of an error, each NE reaches a decision on who the guilty party might be.

Class 2 Messages

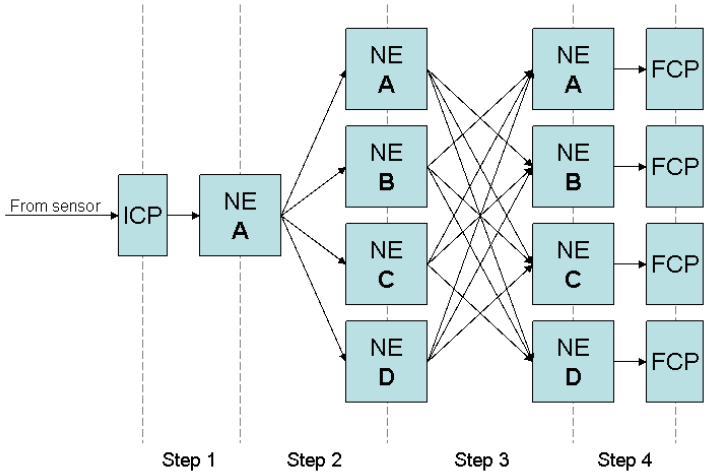

Figure 3-7: A class 1 message requires four steps to travel from an ICP to an FCP. The message is first sent from the ICP to the adjacent NE. Next, the NE initiates a two-step, Byzantine fault tolerant message exchange for the message to reach the other NEs. When the multiple copies of the message have been received by the NEs, each NE votes the message, then forwards it to its adjacent FCP.

Figure 3-8 shows the path of a class 2 message from a set of FCPs to an ICP. A class 2 message requires three hops en route to its destination.

1. The message travels from the FCP to its associated NE.

2. The NE recognizes that the message originated from an FCP, then initiates a class 2 message exchange by sending a copy of the message to all other NEs in the system.

3. Each receiving NE now has a version of the FCP output from all NEs and performs two steps in parallel

   (a) Each NE checks whether it is responsible for a member of the destination VG indicated is the message. If so, the NE determines the correct message by comparing its multiple copies, then forwards the message to the appropriate ICP.

   (b) Each NE also reflects the multiple versions of the FCP output to all other NEs so that, it the event of an inconsistency, all NEs can determine the guilty node.

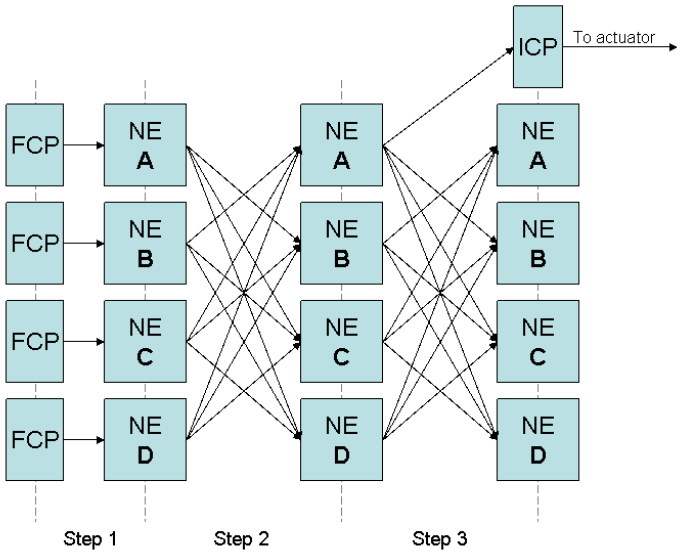

Figure 3-8: A class 2 message requires three steps to travel from an FCP to an ICP. The message is first sent from the FCP to the adjacent NE. Next, the NE sends a copy of the message to all other NEs. Step three involves two parallel operations. First, each NE checks to see if it is responsible for the destination ICP. If it is, it forwards the voted message to that ICP. At the same time, each NE reflects the messages it received in step two to all other NEs. This way, if any of the FCPs send inconsistent output, all the NEs may agree on the guilty party.

3.2.3 Code Structure

The source code is divided into major components: communication libraries, initialization routine, and communication loop. These three parts rely heavily on two third-party libraries, Integrity's OS API and Interpeak's IPCOM/IPLITE lightweight network stack. The Integrity API is particularly important for running multiple tasks and scheduling the execution of those tasks via timers and "alarms." The Interpeak network stack provides a BSD-compliant API for IP/UDP functionality designed for embedded applications.

The communication libraries define the board and virtual process abstraction layers and associated APIs. They allow an NE to send and receive messages to its associated FCP and ICP and to other NEs without having specific knowledge on which physical boards those processors are executing. Currently, the mapping between VPs and physical nodes is hand-

coded into the libraries, but further development would allow those tables to be initialized at startup.

The libraries provide a send/receive mechanism based on fixed-size packets, and allow applications to wait on the arrival of packets using a Unix select-like interface. A user defines a set of NEs and/or FCP/ICPs to wait for a specified (or indeterminate) length of time. The operation will block until a message arrives for any of the defined processors or until the specified time elapses. If the wait operation indicates that a particular processor has messages waiting, a read operation for that processor is guaranteed not to block.

The initialization routine is primarily an implementation of the non-authenticated "Optimal Clock Synchronization" algorithm described in [3]. Its job is to ensure the clocks of the nodes are sufficiently synchronized before the synchronization algorithm described in this thesis takes over maintenance. The implementation consists of two tasks. One task's responsibility is to keep track of time and broadcast "init" packets at the start of each synchronization round. The other task waits for incoming packets, which may arrive before and after what the node believes is the start of the synchronization round. When a packet arrives, the task takes appropriate action, i.e., recording the packet's arrival, sending out an echo packet when appropriate, or adjusting the system clock forward.

Once the initialization routine has completed, control is passed to the communication loop, which manages both the VP and VG communication layers. The communication loop is the heart of the NE, which functions as a fault-tolerant bridge between FCPs and ICPs, maintains synchronization, and controls message sending, receiving and voting. The loop runs as a single thread, performing each responsibility in sequence and repeating every fixed amount of time.

The communication loop repeats the following steps:

1. Prepare and send messages

2. Wait to receive all messages

3. Process and vote appropriate messages

4. Send and receive messages from FCP/ICP

At the beginning of a communication round, the communication loop wakes up, examines its active message table to determine which messages must be sent to other NEs, prepares a buffer containing the messages specific for each NE, and sends them all. The contents of the buffers to each NE are not necessarily the same. For instance, a NE does not need to reflect a message back to the node from which it received the message originally.

After sending, each node waits a pre-specified amount of time to allow messages from trailing nodes to arrive. During this time, the processor is freed for other work to be performed by the CPU, such as executing FCP code.

When the waiting time has elapsed, the NE then checks all the incoming messages for header corruption and adds the new message information to the appropriate locations in the active message table. For example, if the NE receives multiple messages from different NEs containing different versions of the same class 1 message, those versions will be grouped together in the message table. When enough message versions have been received from the other NEs, the versions are compared for inconsistencies. If one version contains errors, they are out-voted by the consistent messages, and thus the errors will not be propagated to the destination FCP or ICP.

The final task of the NE's communication loop is to handle messages to and from its associated FCP/ICPs. If any voted messages are intended for an FCP or ICP for which the NE is responsible, the NE then forwards the message. If any messages are available from the FCP/ICP to be received by the NE, the NE reads them and adds them to the message table in preparation for the next communication cycle. The NE then sleeps until the next cycle, allowing other processes on the node to receive CPU time.

Message Table

The description of the communication loop mentioned the active message table. Each NE maintains a table of active messages is the system. An active message is one that has been sent by an FCP or ICP, but has not yet been delivered to the recipient FCP or ICP. A message is created in the NE's table when it is received from an FCP/ICP or when it is received from another NE during a first-round broadcast for a class 1 or class 2 message. The message is deleted when it either delivered to a recipient FCP/ICP or when the NE recognizes it is not responsible for the destination processor. The table may be thought of as part of the Virtual Group layer, since the table keeps track of related class 1 and class 2 messages for voting purposes.

Other Capabilities

In addition to the SBFTC functionality, the system also includes meta-capabilities to assist in development, debugging, and demonstration. First, the system provides a logging capability that allows debugging text to be stored to a memory buffer rather than a debugging console. This allows debugging and status information to be collected without significantly impacting the timing of the code, particularly the communication loop, and then replayed at the developer's convenience. Text printed directly to the console significantly delays the execution of code, which causes unpredictable behavior between nodes expecting messages within time windows.

In order to support delayed log printing and other on-demand services, an administration interface provides the ability for a user to send UDP packets to specific nodes that instruct them to perform actions like dumping the debug log. If a task wishes to accept external commands, it may register a callback function and a command header such as

“LOG”. The administrator interface then listens on a dedicated port for incoming packets. If a packet arrives in which contents follow the pattern “LOG *”, the specified callback function is invoked with the packet’s contents as an argument.

The administration interface is also used to allow a user to inject faults into the system at will for testing and demonstration purposes.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Synchronization Protocol

Each of the Network Elements (NE) in the SBFTC must maintain its internal clock at approximately the same time to ensure proper functioning of the system. Periodic adjustments of the local clocks are necessary, since all clocks drift relative to one another, and eventually the skew will prevent the boards from behaving in a consistent, predictable manner. The primary reason for keeping the clocks on each board synchronized is to ensure predictability of the communication system and to ease the implementation of the fault tolerant network.

In many multi-node applications, it is often possible and convenient to directly wire all the nodes with the same electrical clock signal. However, in a highly available system, using a common clock signal is impossible because it introduces a single point of failure. If the clock faults, the entire system goes down.

Fault tolerant solutions using custom hardware are able to solve the problem of synchronization by comparing redundant clock signals emanating from each node and adjusting them via specialized hardware. An SBFTC running on COTS hardware cannot rely on such a solution, since low-cost hardware does not include mechanisms for reliable electrical clock synchronization. Instead, each node must maintain a synchronized logical clock

that should keep approximately the same time as the logical clocks on all other nodes. A node maintains its logical clock through two values, an absolute time and a delta. The absolute time is incremented according to the hardware clock. The delta may be adjusted programmatically. The logical clock time is calculated by summing the absolute value and the delta.

Operating systems provide such a clock. However, the challenge exists in adjusting the delta component on all nodes correctly. Since no hardware support exists to synchronize logical clocks, the software is responsible for determining when clock adjustments are necessary by passing messages over a standard communication medium.

The following sections describe a logical clock synchronization algorithm developed for the SBFTC as part of this thesis work. The first section introduces the concepts and intuition behind the algorithm. The next section describes the notation used in this section as well as the assumptions the algorithm depends on. The third section presents a failstop (non-Byzantine resilient) algorithm. The fourth section extends the algorithm for Byzantine resilience. The Byzantine resilient algorithm is introduced, but a complete mathematical analysis has not been completed.

4.1 Motivation and Description

The fault-tolerant network is a bottleneck resource in the SBFTC. In order to make behavior of the system predictable and useful, the SBFTC must guarantee a user application that any data sent across the network will have a *maximum* latency across the network. If that guarantee is violated, time-critical operations may cease to function correctly.

The smallest maximum latency the SBFTC can promise to user applications is the latency of a standard message across the network *plus* the maximum time that message may have to wait for other, higher priority operations to finish using the network. If the sys-

tem must reserve the network periodically for system-level operations like clock synchronization, applications can never be guaranteed latencies less than the time that operation consumes.

Given the importance of reducing required network overhead, the intent of this algorithm is to allow the nodes of the system to maintain a synchronized logical clock while reserving the network for the smallest time intervals possible. While the Byzantine resilient algorithm actually requires a large amount of data to be exchanged between nodes, it allows individual communication rounds to be separated by time, allowing critical messages to be sent in between.

Another motivation in the design of these algorithms is the requirement that clocks never be set backward. In real-time systems, where tasks are scheduled via the system clock and messages may be timestamped, adjusting a clock backwards can result in inconsistent operation unless great care is taken. These algorithms avoid that pitfall by always setting clocks forward or not at all.

Defining Clock Synchronization

Synchronization does not imply that the logical clock is exactly the same on every node at all times. Such perfect synchronization is not possible at the software level. This is not to say that the logical clocks cannot be perfectly synchronized at some *instant*, but maintaining constant perfect synchronization over a time interval is not possible without the high granularity of control possible at the hardware level.

Synchronization of a set of logical clocks is instead defined in a more relaxed way. The logical clocks of a set of k correctly functioning nodes $N = \{n_1, n_2, ..., n_k\}$ are considered

synchronized over a time interval from t_0 to t_1 if

$$\exists\, D_{max} \text{ s.t. } \forall t \in [t_0, t_1],\ \forall n_i, n_j \in N,\ \left|C_i(t) - C_j(t)\right| \leq D_{max}\ ,$$

where $C_i(t)$ is the value of node i's logical clock at real time t and D_{max} is a constant.

In other words, the logical clocks of the nodes are considered synchronized if the difference between any two logical clocks is at most D_{max}. This relaxed definition of synchronization is necessary because it is impossible to achieve perfect synchronization.

Equally important, the relaxed constraint allows time to pass and clocks to drift between synchronization rounds. To describe this notion more precisely, begin with the following definitions:

1. Let A_{sync} be a synchronization algorithm that runs periodically.
2. Let the drift rate between the physical clocks of two nodes be bounded by a constant dr.
3. Let D_{max} be the minimum synchronization (i.e. a maximum difference between logical clocks) required for an application.
4. Let $D_{sync} \leq D_{max}$ be the minimum synchronization achieved by A_{sync}.

Then, the amount of time $dt_{between}$ allowed between points of maximum synchronizations is bounded by

$$dt_{between} \leq \frac{D_{max} - D_{sync}}{dr} - \epsilon,$$

where ϵ is a constant. The role of ϵ is to leave opportunity for mid-execution clock adjustments that may temporarily increase clock skew more than natural drift would.

The algorithms presented below run at well-defined intervals as described above. At the appropriate time according to its logical clock, each node sends a message to every other

node, and each receiving node takes note of the time that each message arrives (according to its own logical clock). Then, given that

- All nodes send their messages at what they believe is the correct time,
- The network has a known minimum and maximum latency,

each node can infer the local logical clock value at each of the other nodes and update its clock appropriately.

As will be shown in the following sections, the achievable D_{sync} value of the synchronization algorithms depends on properties of the network and the drift rate between nodes.

4.2 Definitions and Assumptions

This section presents definitions and assumptions used in the characterization of the synchronization algorithms presented in the following sections.

Node: An independent processing unit with an on-board timekeeper and a method of communication with all other nodes.

Wall clock: A timekeeper that maintains an absolute reference time. This time is not available to the nodes, but as an observer, one is aware of it.

Local clock: A timekeeper belonging to each node. Each local clock keeps time independently from the others. Different local clocks may read different times at the same wall clock time.

Clock drift: The tendency for local clocks on different nodes to run at slightly different rates.

Local clock drift from wall time is assumed to be bounded by a known constant $\boldsymbol{\rho} > 0$. Let $\boldsymbol{C_i(t)}$ be the local clock reading of node i at time t. Then,

$$(1+\rho)^{-1}(t_2 - t_1) \leq C_i(t_2) - C_i(t_1) \leq (1+\rho)(t_2 - t_1). \tag{4.1}$$

The above may be interpreted as, given two times, t_1 and t_2, the amount of logical clock time that could elapse on a node during that period is bounded by linear envelope.

The above may also be rewritten as

$$(1+\rho)^{-1}\left(C_i(t_2) - C_i(t_1)\right) \leq t_2 - t_1 \leq (1+\rho)\left(C_i(t_2) - C_i(t_1)\right). \tag{4.2}$$

Similarly, this may be read as, given two logical clock times, $C_i(t_1)$ and $C_i(t_2)$, the amount of real time that could elapse during that period is bounded by a linear envelope.

Note that the local clock drift rate between any two nodes is bounded by

$$dr = (1+\rho) - (1+\rho)^{-1} = \frac{\rho(2+\rho)}{1+\rho}.$$

It should be noted that, intuitively, the lower drift bounds might more correctly be $(1-\rho)(t_2 - t_1) \leq C_i(t_2) - C_i(t_1)$. However, the type of bound given above is more convenient, and is consistent with other literature, including [9].

Let wall times be denoted by a lowercase t and local clock times be denoted by an uppercase T.

Let $\boldsymbol{C_i(t)}$ be the value of the local clock of node i at time t.

Let $\boldsymbol{D_i(t)}$ be the difference between the local clock of node i and time t at time t. In other words,

$$C_i(t) = t + D_i(t).$$

Let $\boldsymbol{D_{ij}(t)}$ be the difference between the local clocks of nodes i and j at time t. This can be expressed as

$$D_{ij}(t) = D_i(t) - D_j(t) = C_i(t) - C_j(t).$$

Let $\boldsymbol{D_{max}}$ denote the maximum value of $D_{ij}(t)$ acceptable for an application. More precisely,

$$\forall \text{ nodes } i, j, \forall \text{ times } t, \left|D_{ij}(t)\right| \leq D_{max}.$$

Let $\boldsymbol{\left[D_{ij}\right]_i}$ be the difference between the local clocks on nodes i and j, *as calculated by node i.* Note that $\left[D_{ij}\right]_i$ is not a function of t. It represents a single value calculated by node i.

Let $\boldsymbol{T_{send}}$ be the local time at which all nodes are scheduled to send a message to every other node.

Let $\boldsymbol{t_{send_i}}$ be the time at which node i actually sends its messages to the other nodes. Note that T_{send} may correspond to different t_{send_i}'s for different nodes. Note that

$$C_i(t_{send_i}) = T_{send}.$$

Let $\boldsymbol{m_{ij}}$ represent a message sent from node i to node j.

Let $\boldsymbol{t_{m_{ij}}}$ be the time at which m_{ij} is received by node j.

Let $\boldsymbol{T_{m_{ij}}}$ be the local time on node j at which m_{ij} was received by node j.

Let $\boldsymbol{\ell_{m_{ij}}}$ be the time it takes for m_{ij} to travel from node i to node j. Note the relationship

$$t_{m_{ij}} = t_{send_i} + \ell_{m_{ij}}.$$

,

The communication network is assumed to deliver a message in a bounded window of time. Thus,

Let $\boldsymbol{\ell_{max}}$ be the maximum possible value of any $\ell_{m_{ij}}$.

Let $\boldsymbol{\ell_{min}}$ be the minimum possible value of any $\ell_{m_{ij}}$. In other words,

$$\forall \text{ nodes } i, j, \quad \ell_{min} \leq \ell_{m_{ij}} \leq \ell_{max}.$$

Let $\boldsymbol{\ell_{assume}}$ represent a constant such that $\ell_{min} \leq \ell_{assume} \leq \ell_{max}$. It may be interpreted as a best guess for $\ell_{m_{ij}}$. Its optimal value will be calculated as part of the analysis.

Finally, correct functioning of this algorithm requires that all local clocks are initialized such that they are synchronized to within some known value. No initialization algorithm based on the discussed algorithm is provided in this paper. It is left for future work.

4.3 Failstop Algorithm

This section presents the characterization of the non-Byzantine resilient synchronization algorithm. First, the algorithm is analyzed for two nodes. Then the analysis is extended to include three nodes.

The algorithm runs symmetrically on all participating nodes. Algorithm 4.3.1 defines the steps of a single round of the failstop algorithm. The algorithm may be broken into two processes:

1. **Receive:** The node maintains a process at all times that timestamps packets when they arrive.

2. **Send, Wait, and Adjust:** Upon a predetermined time, T_{send}, the node sends a message to all other nodes. The node then spends a predetermined length of time τ_{wait} waiting for the arrival of messages from other, slower nodes. From the arrival times of each message, the node estimates the largest difference between its local clock and other local clocks. If the node estimates that its own local clock is ahead of all others, it does not adjust its local clock. If it estimates that its local clock is behind the clocks of at least one other node, it adjusts its own clock to match that of the node estimated to be most ahead.

In the next round of synchronization, T_{send} takes on another value agreed upon by all participating nodes. Typically, the value of T_{send} at round i, $T_{send}^{(i)}$, is defined by $T_{send}^{(i)} = T_{send}^{(0)} + iP$, where P is some constant.

Figure 4-1 depicts a timing diagram of the interaction between two nodes executing algorithm 4.3.1 between which clocks are initially skewed by an amount Δ. Note that although each node sends a message at what it believes is T_{send}, the messages are actually sent at different times.

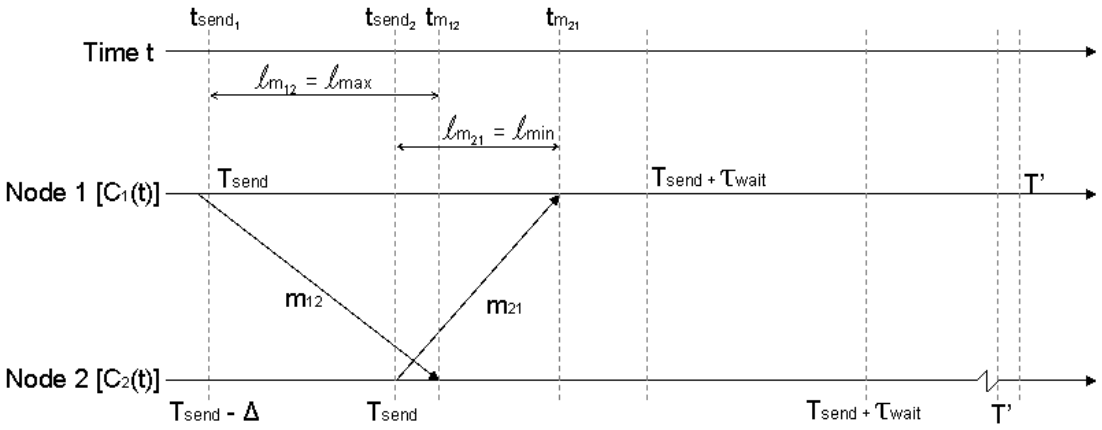

Figure 4-1: An example timing diagram of the interaction between two nodes. Not to scale. The top horizontal line represents wall time. The lower two represent the progression of local clock time for nodes **1** and **2**. The two nodes send a message at what each believes to be Tsend. Note that at time t_{send_1}, node **1**'s clock reads T_{send} and node **2**'s clock reads $T_{send} - \Delta$, indicating that $D_{\mathbf{12}}(t_{send_1}) = \Delta$. Each message may take a different amount of time to reach its destination. In this case, $m_{\mathbf{12}}$ takes the longest time possible and $m_{\mathbf{21}}$ takes the shortest time possible. Each node counts out τ_{wait} after sending its message, then checks to see if it should adjust its clock forward. Each node infers the other's relative clock skew based on the arrival time of the received message. In this case, node **1** realizes it is first and does not adjust its clock. Node **2** adjusts its clock forward to match **1**'s as closely as possible, in this case a little too much, since node **2** reaches T' ahead of **1**.

Algorithm 4.3.1: FAILSTOP-SYNC()

Let $N = \{n_1 ... n_k\}$ be a set of k participating nodes.
Let $i \in \{1...k\}$ be the number of the node executing the instance of the algorithm.
τ_{wait} will be derived during later analysis

global $timestamps[k] \leftarrow \{null, ..., null\}$

process RECEIVE-MESSAGES()
while $C_i(t) < T_{send} + \tau_{wait}$
do { **if** m_{ji} received
then $timestamps[j]$ = CURRENT-TIME()

process SEND-AND-ADJUST()
local $deltas[k] \leftarrow \{0, ...0, \}$

if $C_i(t) = T_{send}$
then {
for $j \leftarrow 1$ **to** k
do { **if** $j \neq i$
then Send message m_{ij} to n_j.

while $C_i(t) < T_{send} + \tau_{wait}$
do { Nothing

for $j \leftarrow 1$ **to** k
do { **if** $j \neq i$
then $deltas[j] \leftarrow$ CALCULATE-DELTA($timestamps[j]$)

SET-TIME(CURRENT-TIME() − MIN($deltas$))

procedure CALCULATE-DELTA($timestamp$)
$delta \leftarrow timestamp - (T_{send} + \ell_{assume})$
return ($delta$)

Figure 4-2: Failstop synchronization algorithm. Process Receive-Messages and process Send-And-Adjust run separately. The first process timestamps messages as they arrive from other nodes. The second sends messages, waits for all messages to arrive, then adjusts the clock forward if necessary.

4.3.1 Characterization of Algorithm for Two Nodes

This section provides the mathematical characterization of the failstop synchronization algorithm and demonstrates that it is correct.

The analysis is organized as follows:

1. First, bounds on the nodes' estimation of the other's clock skew are determined in terms the initial relative clock skew between the nodes.

2. Next, the estimation bounds from the previous result are used to determine the clock adjustment behavior under all possible initial relative clock skews.

3. After understanding how nodes adjust their clocks, bounds are derived for the maximum clock skew possible following the execution of the algorithm.

4. Next, an expression is derived for the actual maximum skew possible at any point during the algorithm in term of the elapsed time between synchronizations.

5. Finally, lower bounds are derived on how frequently the algorithm may be executed, thereby allowing the actual maximum synchronization to be calculated using the results from step 4.

Determining each node's perception of local clock difference

To begin, consider two nodes, **1** and **2**, performing algorithm 4.3.1. The analysis of the two nodes will not generalize to the case of three or more nodes, but it is described first because it is the most straightforward to analyze.

Without loss of generality, the local clock of node **1** is defined to be equal or ahead of **2**'s at the start of the algorithm. In other words,

$$D_{\mathbf{12}}(t_{start}) \geq 0.$$

The first step of analysis is to determine the bounds of **1** and **2**'s perception of the other nodes' local clocks in terms of the relative skew between them at the start of the algorithm.

The start of the algorithm is defined to be the time at which the first node sends a message. Since node **1**'s clock is defined to be ahead of **2**'s at the start of the algorithm, by definition,

$$D_{\mathbf{12}}(t_{start}) = D_{\mathbf{12}}(t_{send_{\mathbf{1}}}).$$

First consider how node **2** perceives the local clock of node **1**. Begin by recalling the formula that the nodes use to estimate the difference between their local clocks based on message arrival time,

$$[D_{ij}]_i = C_i(t_{m_{ji}}) - (T_{send} + \ell_{assume}).$$

Therefore, **2** predicts the difference between **1**'s and its own local clocks by

$$[D_{\mathbf{21}}]_{\mathbf{2}} = C_{\mathbf{2}}(t_{m_{\mathbf{12}}}) - (T_{send} + \ell_{assume}),$$

which may be also expressed as

$$[D_{\mathbf{21}}]_{\mathbf{2}} = t_{m_{\mathbf{12}}} + D_{\mathbf{2}}(t_{m_{\mathbf{12}}}) - (T_{send} + \ell_{assume}). \tag{4.3}$$

Next, consider amount of time that will elapse on **2**'s local clock from when **1** sends the message $m_{\mathbf{12}}$ at time $t_{send_{\mathbf{1}}}$ to when **2** receives the message at time $t_{m_{\mathbf{12}}}$. Equation (4.1) provides bounds on the elapsed time:

$$\begin{aligned} C_2(t_{m_{12}}) - C_2(t_{send_1}) &\leq (1+\rho)(t_{m_{12}} - t_{send_1}) \\ C_2(t_{m_{12}}) - C_2(t_{send_1}) &\geq (1+\rho)^{-1}(t_{m_{12}} - t_{send_1}). \end{aligned}$$

Since $C_i(t) = t + D_i(t)$, the above can be rewritten as

$$\begin{aligned} t_{m_{12}} + D_2(t_{m_{12}}) - t_{send_1} - D_2(t_{send_1}) &\leq (1+\rho)(t_{m_{12}} - t_{send_1}) \\ t_{m_{12}} + D_2(t_{m_{12}}) - t_{send_1} - D_2(t_{send_1}) &\geq (1+\rho)^{-1}(t_{m_{12}} - t_{send_1}). \end{aligned}$$

Isolating $D_2(t_{m_{12}})$ yields

$$\begin{aligned} D_2(t_{m_{12}}) &\leq (1+\rho)(t_{m_{12}} - t_{send_1}) - (t_{m_{12}} - t_{send_1}) + D_2(t_{send_1}) \\ D_2(t_{m_{12}}) &\geq (1+\rho)^{-1}(t_{m_{12}} - t_{send_1}) - (t_{m_{12}} - t_{send_1}) + D_2(t_{send_1}) \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_2(t_{m_{12}}) &\leq \rho(t_{m_{12}} - t_{send_1}) + D_2(t_{send_1}) \\ D_2(t_{m_{12}}) &\geq -\left(\frac{\rho}{1+\rho}\right)(t_{m_{12}} - t_{send_1}) + D_2(t_{send_1}). \end{aligned}$$

Recognizing that $t_{m_{12}} - t_{send_1} = \ell_{m_{12}}$ and that $\ell_{min} \leq \ell_{m_{12}} \leq \ell_{max}$ gives

$$\begin{aligned} D_2(t_{m_{12}}) &\leq (\rho)\ell_{max} + D_2(t_{send_1}) \\ D_2(t_{m_{12}}) &\geq -\left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_2(t_{send_1}). \end{aligned}$$

From these inequalities, bounds can now be found for $[D_{21}]_2$ using equation (4.3):

$$
\begin{aligned}
[D_{21}]_2 &\leq t_{m_{12}} + (\rho)\ell_{max} + D_2(t_{send_1}) - (T_{send} + \ell_{assume}) \\
[D_{21}]_2 &\geq t_{m_{12}} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_2(t_{send_1}) - (T_{send} + \ell_{assume}).
\end{aligned}
$$

Finally, by recognizing that $T_{send} = C_1(t_{send_1}) = t_{send_1} + D_1(t_{send_1})$, the following bounds are achieved:

$$
\begin{aligned}
[D_{21}]_2 &\leq t_{m_{12}} + (\rho)\ell_{max} + D_2(t_{send_1}) - (t_{send_1} + D_1(t_{send_1}) + \ell_{assume}) \\
[D_{21}]_2 &\geq t_{m_{12}} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_2(t_{send_1}) - (t_{send_1} + D_1(t_{send_1}) + \ell_{assume})
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad [D_{21}]_2 &\leq (t_{m_{12}} - t_{send_1}) + (\rho)\ell_{max} + (D_2(t_{send_1}) - D_1(t_{send_1})) - \ell_{assume} \\
[D_{21}]_2 &\geq (t_{m_{12}} - t_{send_1}) - \left(\frac{\rho}{1+\rho}\right)\ell_{max} + (D_2(t_{send_1}) - D_1(t_{send_1})) - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad [D_{21}]_2 &\leq (\ell_{m_{12}}) + (\rho)\ell_{max} + (D_{21}(t_{send_1})) - \ell_{assume} \\
[D_{21}]_2 &\geq (\ell_{m_{12}}) - \left(\frac{\rho}{1+\rho}\right)\ell_{max} + (D_{21}(t_{send_1})) - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad [D_{21}]_2 &\leq \ell_{max} + (\rho)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume} \\
[D_{21}]_2 &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume},
\end{aligned}
$$

which finally simplifies to

$$
\begin{aligned}
[D_{21}]_2 &\leq (1+\rho)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume} \\
[D_{21}]_2 &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume}.
\end{aligned} \tag{4.4}
$$

In the next step, the bounds of **1**'s perception of **2**'s local clock are determined. Begin by recognizing that

$$[D_{12}]_1 = C_1(t_{m_{21}}) - (T_{send} + \ell_{assume}). \tag{4.5}$$

The challenge here is to find $C_1(t_{m_{21}})$ in terms of the local clock difference at the start of the algorithm, namely $D_{12}(t_{send_1})$. Begin by determining the bounds of the elapsed time on **1**'s local clock in terms of real time. Equation (4.1) gives the following:

$$\begin{aligned} C_1(t_{m_{21}}) - C_1(t_{send_1}) &\leq (1+\rho)(t_{m_{21}} - t_{send_1}) \\ C_1(t_{m_{21}}) - C_1(t_{send_1}) &\geq (1+\rho)^{-1}(t_{m_{21}} - t_{send_1}). \end{aligned} \tag{4.6}$$

Since $t_{m_{21}} = t_{send_2} + \ell_{m_{21}}$ and $\ell_{min} \leq \ell_{m_{21}} \leq \ell_{max}$, bounds on $t_{m_{21}}$ are as follows

$$\begin{aligned} t_{m_{21}} &\leq t_{send_2} + \ell_{max} \\ t_{m_{21}} &\geq t_{send_2} + \ell_{min}. \end{aligned} \tag{4.7}$$

Bounds on t_{send_2} can be determined using equation (4.2):

$$\begin{aligned} t_{send_2} - t_{send_1} &\leq (1+\rho)(C_2(t_{send_2}) - C_2(t_{send_1})) \\ t_{send_2} - t_{send_1} &\geq (1+\rho)^{-1}(C_2(t_{send_2}) - C_2(t_{send_1})). \end{aligned}$$

Recognizing that $C_2(t_{send_2}) = T_{send} = C_1(t_{send_1})$ gives

$$
\begin{aligned}
t_{send_2} - t_{send_1} &\leq (1+\rho)(C_1(t_{send_1}) - C_2(t_{send_1})) \\
t_{send_2} - t_{send_1} &\geq (1+\rho)^{-1}(C_1(t_{send_1}) - C_2(t_{send_1}))
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad t_{send_2} - t_{send_1} &\leq (1+\rho)D_{12}(t_{send_1}) \\
t_{send_2} - t_{send_1} &\geq (1+\rho)^{-1}D_{12}(t_{send_1})
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad t_{send_2} &\leq (1+\rho)D_{12}(t_{send_1}) + t_{send_1} \\
t_{send_2} &\geq (1+\rho)^{-1}D_{12}(t_{send_1}) + t_{send_1}.
\end{aligned}
$$

Substituting for t_{send_2} in equation (4.7) gives

$$
\begin{aligned}
t_{m_{21}} &\leq (1+\rho)D_{12}(t_{send_1}) + t_{send_1} + \ell_{max} \\
t_{m_{21}} &\geq (1+\rho)^{-1}D_{12}(t_{send_1}) + t_{send_1} + \ell_{min}.
\end{aligned}
$$

Next, referring back to the bounds on $C_1(t_{m_{21}}) - C_1(t_{send_1})$ given by equation (4.6) and substituting for $t_{m_{21}}$ yields

$$
\begin{aligned}
C_1(t_{m_{21}}) - C_1(t_{send_1}) &\leq (1+\rho)((1+\rho)D_{12}(t_{send_1}) + t_{send_1} + \ell_{max} - t_{send_1}) \\
C_1(t_{m_{21}}) - C_1(t_{send_1}) &\geq (1+\rho)^{-1}((1+\rho)^{-1}D_{12}(t_{send_1}) + t_{send_1} + \ell_{min} - t_{send_1})
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad C_1(t_{m_{21}}) - C_1(t_{send_1}) &\leq (1+\rho)^2 D_{12}(t_{send_1}) + (1+\rho)\ell_{max} \\
C_1(t_{m_{21}}) - C_1(t_{send_1}) &\geq (1+\rho)^{-2}D_{12}(t_{send_1}) + (1+\rho)^{-1}\ell_{min}.
\end{aligned}
$$

Finally, since $C_1(t_{send_1}) = T_{send}$, the above bounds can be directly substituted into equa-

tion (4.5), determining the bounds on $[D_{\mathbf{12}}]_{\mathbf{1}}$ in terms of $D_{\mathbf{12}}(t_{send_1})$:

$$\begin{aligned} [D_{\mathbf{12}}]_{\mathbf{1}} &\leq (1+\rho)^2 D_{\mathbf{12}}(t_{send_1}) + (1+\rho)\ell_{max} - \ell_{assume} \\ [D_{\mathbf{12}}]_{\mathbf{1}} &\geq (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}. \end{aligned} \tag{4.8}$$

Determining limits of behavior

Now that the bounds for $[D_{\mathbf{12}}]_{\mathbf{1}}$ and $[D_{\mathbf{21}}]_{\mathbf{2}}$ have been determined in terms of the initial clock difference, it is possible to evaluate how nodes will adjust their local clocks based on their approximation of their distance from the other node.

The amount that a node i adjusts its clock is a function of $[D_{ij}]_i$, its approximation of its own clock's distance from j's clock, but the function is not smooth over all values of $[D_{ij}]_i$, but rather a piecewise function of the form

$$adjust([D_{ij}]_i) = \begin{cases} -[D_{ij}]_i & , [D_{ij}]_i < 0 \\ 0 & , [D_{ij}]_i \geq 0 \end{cases}.$$

The piecewise nature of the adjustment function requires the behavior analysis of the synchronization to be applied in a piecewise manner as well, with discontinuities existing when either $[D_{ij}]_i = 0$ or $[D_{ji}]_j = 0$.

In other words, the behavior of the system depends on whether one, both, or neither node adjusts its local clock during the algorithm. Different equations will bound the achievable synchronization for each case.

Returning to nodes **1** and **2**, the following questions are considered:

1. Under what conditions is $[D_{\mathbf{12}}]_{\mathbf{1}} \geq 0$ possible?

2. Under what conditions is $[D_{12}]_1 < 0$ possible?

3. Under what conditions is $[D_{21}]_2 \geq 0$ possible?

4. Under what conditions is $[D_{21}]_2 < 0$ possible?

The conditions for all cases are expressed in terms of $D_{12}(t_{send_i})$:

$[\mathbf{D_{12}}]_1 \geq 0$ possible: $[D_{12}]_1 \geq 0$ is possible when the upper bound of $[D_{12}]_1$ is positive. Equation (4.8) gives

$$\begin{aligned} (1+\rho)^2 D_{12}(t_{send_1}) + (1+\rho)\ell_{max} - \ell_{assume} &\geq 0 \\ \Rightarrow \qquad D_{12}(t_{send_1}) &\geq (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{max}. \end{aligned}$$

Note that, by definition, $\rho > 0$, $\ell_{assume} \leq \ell_{max}$, and $D_{12}(t_{send_1}) \geq 0$, thus the above inequality is always true. Therefore, it is always possible that node **1** estimates its local clock to be ahead of **2**'s. This makes sense, since **1**'s clock is defined to be the same or ahead of **2**'s at the start of the algorithm.

$[\mathbf{D_{12}}]_1 < 0$ possible: $[D_{12}]_1 < 0$ is possible when the lower bound of $[D_{12}]_1$ is negative. Equation (4.8) gives

$$\begin{aligned} (1+\rho)^{-2} D_{12}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume} &< 0 \\ \Rightarrow \qquad D_{12}(t_{send_1}) &< (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}. \end{aligned}$$

The above condition can be true or false depending on the values of ρ, ℓ_{max} and ℓ_{assume}.

$[\mathbf{D_{21}}]_2 \geq 0$ possible: Equation (4.4) gives

$$(1+\rho)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume} \geq 0$$

$$\Rightarrow \qquad D_{\mathbf{12}}(t_{send_1}) \;\leq\; (1+\rho)\ell_{max} - \ell_{assume}.$$

$[\mathbf{D_{21}}]_2 < \mathbf{0}$ possible: Equation (4.4) gives

$$\begin{aligned} \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} &< 0 \\ \Rightarrow \qquad D_{\mathbf{12}}(t_{send_1}) &> \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume}. \end{aligned}$$

Since $\ell_{assume} > \ell_{min}$, this inequality is always true.

With these limits established, limits can now be found for each of the behavior cases by looking at the intersection of the limits.

$[\mathbf{D_{12}}]_1 \geq \mathbf{0} \wedge [\mathbf{D_{21}}]_2 < \mathbf{0}$ possible: The case of only node **2** adjusting its clock is always possible, i.e.,

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\geq (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{max} \\ D_{\mathbf{12}}(t_{send_1}) &> \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} \end{aligned}$$

$$\Rightarrow \quad D_{\mathbf{12}}(t_{send_i}) > 0.$$

$[\mathbf{D_{12}}]_1 < \mathbf{0} \wedge [\mathbf{D_{21}}]_2 < \mathbf{0}$ possible: The case of both nodes **1** and **2** adjusting their clocks is possible when

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &< (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} \\ D_{\mathbf{12}}(t_{send_1}) &> \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_{\mathbf{12}}(t_{send_1}) &< (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} \\ D_{\mathbf{12}}(t_{send_1}) &> 0 \end{aligned}$$

$[\mathbf{D_{12}}]_1 < \mathbf{0} \wedge [\mathbf{D_{21}}]_2 \geq \mathbf{0}$ **possible:** The case of only node **1** adjusting its clock is possible when

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &< (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} \\ D_{\mathbf{12}}(t_{send_1}) &\leq (1+\rho)\ell_{max} - \ell_{assume} \end{aligned}$$

$[\mathbf{D_{12}}]_1 \geq \mathbf{0} \wedge [\mathbf{D_{21}}]_2 \geq \mathbf{0}$ **possible:** Finally, the case of neither node adjusting its clock is possible when

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\geq (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{max} \\ D_{\mathbf{12}}(t_{send_1}) &\leq (1+\rho)\ell_{max} - \ell_{assume} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_{\mathbf{12}}(t_{send_1}) &\geq 0 \\ D_{\mathbf{12}}(t_{send_1}) &\leq (1+\rho)\ell_{max} - \ell_{assume} \end{aligned}$$

From these results, the range of $D_{\mathbf{12}}(t_{send_1})$ can be broken up into regions where different clock adjustment behavior is known to occur. Analysis can then be performed for each region independently, then compared. The regions depend on the value of ℓ_{assume}. Let ℓ_{mid} be defined as the value of ℓ_{assume} when $(1+\rho)\ell_{max} - \ell_{assume} = (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}$. Then,

$$\ell_{mid} = ((1+\rho)\ell_{max} + (1+\rho)\ell_{min})\left(1 + (1+\rho)^2\right)^{-1}.$$

If

$$\ell_{assume} > \ell_{mid},$$

then

$$(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} > (1+\rho)\ell_{max} - \ell_{assume},$$

and the possible node behavior in different regions are as follows:

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] : \mathbf{1} \text{ sets, } \mathbf{2} \text{ sets, both set, neither sets} \\ &\in [(1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}] : \mathbf{2} \text{ sets, both set} \\ &\in [(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty) : \mathbf{2} \text{ sets.} \end{aligned}$$

Alternatively, if

$$\ell_{assume} < \ell_{mid},$$

then

$$(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} < (1+\rho)\ell_{max} - \ell_{assume},$$

and the possible node behavior in different regions are instead

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}] : \mathbf{1} \text{ sets, } \mathbf{2} \text{ sets, both set, neither sets} \\ &\in [(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}] : \mathbf{2} \text{ sets, neither sets} \\ &\in [(1+\rho)\ell_{max} - \ell_{assume}, \infty) : \mathbf{2} \text{ sets.} \end{aligned}$$

Determining limits of synchronization

The maximum synchronization achievable through the algorithm depends not only on environment variables like ℓ_{max}, ℓ_{min}, and ρ, but also on how frequently the synchronization

algorithm can run. Given the synchronization of the nodes $D_{\mathbf{12}}(t_{start})$ at the start of a round and the maximum time $dt_{between}$ between rounds, the synchronization of the nodes at the start of the next round, $D_{\mathbf{12}}(t^{(2)}_{start})$, is bounded by

$$\begin{aligned} D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{start}) + \Delta_{sync_{max}} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{start}) + \Delta_{sync_{min}} - dr \cdot dt_{between}, \end{aligned} \tag{4.9}$$

where $\Delta_{sync_{min}}$ and $\Delta_{sync_{max}}$ are the minimum and maximum change in clock skew due to adjustment in a single round, and $dr \cdot dt_{between}$ is the maximum drift between two local clocks possible between the start of two rounds. Note $\Delta_{sync_{min}}$ and $\Delta_{sync_{max}}$ may be broken down further into

$$\begin{aligned} \Delta_{sync_{min}} &= \Delta_{sync_{min_{\mathbf{1}}}} + \Delta_{sync_{min_{\mathbf{2}}}} \\ \Delta_{sync_{max}} &= \Delta_{sync_{max_{\mathbf{1}}}} + \Delta_{sync_{max_{\mathbf{2}}}}, \end{aligned}$$

where $\Delta_{sync_{min_{\mathbf{1}}}}$ is the change contributed by node **1** and $\Delta_{sync_{max_{\mathbf{2}}}}$ is the change contributed by node **2**. Note that a positive value of $\Delta_{sync_{min}}$ or $\Delta_{sync_{max}}$ indicates that node **1** adjusted its clock forward (generally undesirable) while a negative value means that node **2** adjusted its clock forward (generally desirable).

The previous section determined the boundary cases for how nodes adjust their clocks, so it is now possible to calculate $\Delta_{sync_{min}}$ and $\Delta_{sync_{max}}$ for each of the different adjustment cases in each of the different regions.

Case $\ell_{assume} > \ell_{mid}$**,** $D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$**:** In this case the smallest contribution of node **1** is

$$\begin{aligned}\Delta_{sync_{min_{\mathbf{1}}}} &= -\min\{0, \max\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\ &= -\min\left\{0, (1+\rho)^2 D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)\ell_{max} - \ell_{assume}\right\} \\ &= 0.\end{aligned}$$

The largest contribution of node **1** is

$$\begin{aligned}\Delta_{sync_{max_{\mathbf{1}}}} &= -\min\{0, \min\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\ &= -\min\left\{0, (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\} \\ &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}.\end{aligned}$$

The smallest contribution of node **2** is

$$\begin{aligned}\Delta_{sync_{min_{\mathbf{2}}}} &= \min\{0, \min\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\ &= \min\left\{0, \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}\right\} \\ &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}.\end{aligned}$$

The largest contribution of node **2** is

$$\begin{aligned}\Delta_{sync_{max_{\mathbf{2}}}} &= \min\{0, \max\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\ &= \min\{0, (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}\} \\ &= 0.\end{aligned}$$

Therefore, for the case $\ell_{assume} > \ell_{mid}$, $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$,

$$\begin{aligned} \Delta_{sync_{min}} &= 0 + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\ \Delta_{sync_{max}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + 0 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad \Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\ \Delta_{sync_{max}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}. \end{aligned}$$

Substituting these results into equation (4.9) yields

$$\begin{aligned} D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{send_1}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left(1 - (1+\rho)^{-2}\right) D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}. \end{aligned}$$

Since $D_{\mathbf{12}}(t_{send_1}) \leq (1+\rho)\ell_{max} - \ell_{assume}$, substituting for $D_{\mathbf{12}}(t_{send_1})$ gives

$$\begin{aligned} D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left(1 - (1+\rho)^{-2}\right)((1+\rho)\ell_{max} - \ell_{assume}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}. \end{aligned}$$

Case $\ell_{assume} > \ell_{mid}$, $D_{\mathbf{12}}(t_{send_1}) \in [(1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$: In this case, the only possible behaviors are that only node **2** adjusts its clock or that both nodes **1**

and **2** adjust their clocks.

The smallest contribution of node **1** is

$$\begin{aligned}\Delta_{sync_{min_{\mathbf{1}}}} &= -\min\{0, \max\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\ &= 0.\end{aligned}$$

The largest contribution of node **1** is

$$\begin{aligned}\Delta_{sync_{max_{\mathbf{1}}}} &= -\min\{0, \min\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\ &= -\min\left\{0, (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\}.\end{aligned}$$

The smallest contribution of node **2** is

$$\begin{aligned}\Delta_{sync_{min_{\mathbf{2}}}} &= \min\{0, \min\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\ &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}.\end{aligned}$$

The largest contribution of node **2** is

$$\begin{aligned}\Delta_{sync_{max_{\mathbf{2}}}} &= \min\{0, \max\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\ &= (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}.\end{aligned}$$

Therefore, for the case $\ell_{assume} > \ell_{mid}$, $D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) \in [(1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} -$

$(1+\rho)\ell_{min}]$,

$$\begin{aligned}
\Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\Delta_{sync_{max}} &= -\min\left\{0, (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume}.
\end{aligned}$$

Substituting these results into equation (4.9) yields

$$\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq -\min\left\{0, (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\} \\
&\quad +(1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}.
\end{aligned}$$

The smallest possible value for $D_{\mathbf{12}}(t_{send_{\mathbf{1}}})$ in this case is chosen as a substitution, giving

$$\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq -\min\left\{0, (1+\rho)^{-2}\left((1+\rho)\ell_{max} - \ell_{assume}\right) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\} \\
&\quad +(1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}
\end{aligned}$$

$$\begin{aligned}
\Rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq -\left((1+\rho)^{-2}\left((1+\rho)\ell_{max} - \ell_{assume}\right) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right) \\
&\quad +(1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between},
\end{aligned}$$

which simplifies to

$$
\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left((1+\rho)-(1+\rho)^{-1}\right)\ell_{max} - (1+\rho)^{-1}\ell_{min} + (1+\rho)^{-2}\ell_{assume} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}.
\end{aligned}
$$

Case $\ell_{assume} > \ell_{mid}$**,** $D_{\mathbf{12}}(t_{send_1}) \in [(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}], \infty$**:** In this case, the only possible behavior is that only node **2** adjusts its clock. The limits of adjustment are

$$
\begin{aligned}
\Delta_{sync_{min_{\mathbf{1}}}} &= 0 \\
\Delta_{sync_{max_{\mathbf{1}}}} &= 0 \\
\Delta_{sync_{min_{\mathbf{2}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\
\Delta_{sync_{max_{\mathbf{2}}}} &= (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume},
\end{aligned}
$$

giving

$$
\begin{aligned}
\Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\
\Delta_{sync_{max}} &= (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}
\end{aligned}
$$

and

$$
\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}.
\end{aligned}
$$

The same analysis is now performed for the cases where $\ell_{assume} < \ell_{mid}$.

Case $\ell_{assume} < \ell_{mid}$, $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$:

$$\begin{aligned}
\Delta_{sync_{min_{\mathbf{1}}}} &= -\min\{0, \max\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\
&= -\min\left\{0, (1+\rho)^2 D_{\mathbf{12}}(t_{send_1}) + (1+\rho)\ell_{max} - \ell_{assume}\right\} \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\Delta_{sync_{max_{\mathbf{1}}}} &= -\min\{0, \min\{[D_{\mathbf{12}}]_{\mathbf{1}}\}\} \\
&= -\min\left\{0, (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right\} \\
&= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\Delta_{sync_{min_{\mathbf{2}}}} &= \min\{0, \min\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\
&= \min\left\{0, \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}\right\} \\
&= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\Delta_{sync_{max_{\mathbf{2}}}} &= \min\{0, \max\{[D_{\mathbf{21}}]_{\mathbf{2}}\}\} \\
&= \min\{0, (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}\},
\end{aligned}$$

which gives

$$\begin{aligned}
\Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\
\Delta_{sync_{max}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad + \min\{0, (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}\}
\end{aligned}$$

and

$$\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left(1-(1+\rho)^{-2}\right) D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad + \min\{0, (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}\} + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}.
\end{aligned}$$

Ideally, to find the maximum possible value of $D_{\mathbf{12}}(t^{(2)}_{start})$ under these conditions, the value of $D_{\mathbf{12}}(t_{send_1})$ should be selected such that $(1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} = 0$. Any smaller value would decrease the total via the $\left(1-(1+\rho)^{-2}\right) D_{\mathbf{12}}(t_{send_1})$ term, and any larger value would also decrease the total via the $D_{\mathbf{12}}(t_{send_1})$ within the min function. Solving gives $D_{\mathbf{12}}(t_{send_1}) = (1+\rho)\ell_{max} - \ell_{assume}$.

However, since $\ell_{assume} < \ell_{mid}$, then $(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} < (1+\rho)\ell_{max} - \ell_{assume}$, so the largest value $D_{\mathbf{12}}(t_{send_1})$ may take is $(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}$. Therefore, the limits of $D_{\mathbf{12}}(t^{(2)}_{start})$ become

$$\begin{aligned}
D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left(1-(1+\rho)^{-2}\right)\left((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}\right) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad + dr \cdot dt_{between} \\
D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}
\end{aligned}$$

$$\begin{aligned} \Rightarrow \quad D_{\mathbf{12}}(t_{start}^{(2)}) &\leq (1+\rho)^2 \ell_{assume} - (1+\rho)\ell_{min} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t_{start}^{(2)}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}. \end{aligned}$$

Case $\ell_{assume} < \ell_{mid}$**,** $D_{\mathbf{12}}(t_{send_1}) \in [(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}]$**:** In this case, the possible node behaviors are either **2** adjusts its clock or neither adjusts its clock:

$$\begin{aligned} \Delta_{sync_{min_1}} &= 0 \\ \Delta_{sync_{max_1}} &= 0 \\ \Delta_{sync_{min_2}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\ \Delta_{sync_{max_2}} &= 0. \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\ \Delta_{sync_{max}} &= 0 \end{aligned}$$

and

$$\begin{aligned} D_{\mathbf{12}}(t_{start}^{(2)}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t_{start}^{(2)}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}. \end{aligned}$$

Case $\ell_{assume} < \ell_{mid}$**,** $D_{\mathbf{12}}(t_{send_1}) \in [(1+\rho)\ell_{max} - \ell_{assume}, \infty]$**:** In this case, node **2** adjusts its clock:

$$
\begin{aligned}
\Delta_{sync_{min_1}} &= 0 \\
\Delta_{sync_{max_1}} &= 0 \\
\Delta_{sync_{min_2}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume} \\
\Delta_{sync_{max_2}} &= (1+\rho)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume}.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\Delta_{sync_{min}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume} \\
\Delta_{sync_{max}} &= (1+\rho)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume}
\end{aligned}
$$

and

$$
\begin{aligned}
D_{12}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
D_{12}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}.
\end{aligned}
$$

At this point, three upper bounds and three lower bounds have been derived for the clock skew between nodes at the start of the following synchronization round for each of the conditions $\ell_{assume} > \ell_{mid}$ and $\ell_{assume} < \ell_{mid}$. It is now possible to evaluate the actual upper and lower bounds on $D_{12}(t^{(2)}_{start})$ by comparing the bounds in each case and selecting the most extreme. In addition, since ℓ_{assume} is a tunable parameter, it is possible to minimize the extremes by selecting ℓ_{assume} appropriately.

Note that for all six cases,

$$D_{\mathbf{12}}(t_{start}^{(2)}) \geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between},$$

making the selection of the actual (most extreme) lower bound trivial.

The possible maximum values of $D_{\mathbf{12}}(t_{start}^{(2)})$ when $\ell_{assume} > \ell_{mid}$ are

$$\begin{aligned} D_{\mathbf{12}}(t_{start}^{(2)}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t_{start}^{(2)}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}, \end{aligned}$$

and when $\ell_{assume} < \ell_{mid}$, the possible maximum values are

$$\begin{aligned} D_{\mathbf{12}}(t_{start}^{(2)}) &\leq (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} + dr \cdot dt_{between} \\ D_{\mathbf{12}}(t_{start}^{(2)}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}. \end{aligned}$$

Since each of the bounds is linear with respect to $\ell_{assumed}$, and since for each pair of bounds one is positively sloped and one is negatively sloped, the upper bound is at its minimum when the lines intersect. For both pairs, the intersection occurs at $\ell_{assumed} = \ell_{min}$. In addition, at that value of $\ell_{assumed}$,

$$\begin{aligned} |(1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}| &> \left|\ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between}\right| \Rightarrow \\ (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} &> -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + \ell_{assume} + dr \cdot dt_{between} \Rightarrow \\ (1+\rho)\ell_{max} - \ell_{assume} &> -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + \ell_{assume}. \end{aligned}$$

Therefore, the most extreme bound on $D_{\mathbf{12}}(t_{start}^{(2)})$ is minimized at $\ell_{assume} = \ell_{mid}$ and evaluates to

$$\left|D_{\mathbf{12}}(t_{start}^{(2)})\right| \leq (1+\rho)\ell_{max} - \left(1 + (1+\rho)^2\right)^{-1} ((1+\rho)\ell_{max} + (1+\rho)\ell_{min}) + dr \cdot dt_{between}. \quad (4.10)$$

Determining maximum synchronization

It is tempting to assume that, given the bounds for $D_{\mathbf{12}}(t_{start}^{(2)})$ above, the task of finding the maximum synchronization supported is trivial: simply plug in the values of ℓ_{max}, ℓ_{min}, ρ, dr, and the smallest value of $dt_{between}$ achievable and evaluate. However, to do so would neglect the fact that clocks may continue to drift during the execution of the algorithm before node **2** adjusts its clock forward.

Therefore, determining the actual maximum synchronization achievable requires knowledge of how much time may pass between the start of the algorithm and the time node **2** adjusts its clock.

The interval dt_{set_i} between the beginning of the algorithm and when node i adjusts its clock is given by

$$dt_{set_i} = t_{send_i} + dt_{wait_i} + dt_{calc_i} - t_{start},$$

where t_{send_i} is the time at which node i sends its messages to all other nodes, dt_{wait_i} is the length of time it waits to receive messages from other nodes, dt_{calc_i} is the length of time the node takes to perform subsequent calculations after receiving all messages and before adjusting its clock, and t_{start} is the time at which the algorithm begins. Note that for the running example with nodes **1** and **2**,

$$dt_{set_2} = t_{send_2} + dt_{wait_2} + dt_{calc_2} - t_{send_1}.$$

The term dt_{calc_i} is bounded above by,

$$dt_{calc_i} \leq dt_{calc_{max}},$$

where $dt_{calc_{max}}$ is determined experimentally.

A benchmark to determine the maximum amount of time required for calculations would use local clock readings. In other words, a benchmark would determine $\tau_{calc_{max}}$. By the bounds of clock drift from real time, $\tau_{calc_{max}}$ gives

$$dt_{calc_{max}} \leq (1+\rho)\tau_{calc_{max}}$$

and

$$dt_{calc_i} \leq (1+\rho)\tau_{calc_{max}}.$$

The possible values of dt_{wait_i} must be restricted to values that *guarantee* that the node will receive all messages from all other functioning nodes before continuing on to determine how to adjust its clock. Let $dt_{wait_{req}}$ be the minimum value of dt_{wait_i} that provides such a guarantee. Let the related value $\tau_{wait_{req}}$ be the amount of *local* time a node must count on its own local clock to guarantee the passage of $dt_{wait_{req}}$ in real time.

Note that $\tau_{wait_{req}} \neq dt_{wait_{req}}$ because of local clocks' tendency to drift from real time.

If a node is required to wait an interval $dt_{wait_{req}}$ in real time, it must actually count out $\tau_{wait_{req}} > dt_{wait_{req}}$ units of time on its local clock, since it is possible for a local clock to count faster than real time.

From the bounds on clock drift provided by equation (4.2), the actual amount of real time dt_{wait} that may elapse over a local clock interval of $\tau_{wait_{req}}$ can be bounded as follows:

$$\begin{aligned} dt_{wait} &\leq (1+\rho)\tau_{wait_{req}} \\ dt_{wait} &\geq (1+\rho)^{-1}\tau_{wait_{req}}, \end{aligned}$$

where $\tau_{wait_{req}} = C_i(t_{send_i} + dt_{wait}) - C_i(t_{send_i})$.

The requirement that $dt_{wait} \geq dt_{wait_{req}}$ gives

$$\tau_{wait_{req}} > (1+\rho)dt_{wait_{req}}.$$

Now an expression for $dt_{wait_{req}}$ must be determined. Recall that $dt_{wait_{req}}$ is the length of time a node must wait after it sends its messages to guarantee it receives all messages before continuing. Thus, the value of $dt_{wait_{req}}$ may be expressed as

$$dt_{wait_{req}} = max\left\{t_{send_j} - t_{send_i}\right\} + \ell_{max}, \tag{4.11}$$

where $max\left\{t_{send_j} - t_{send_i}\right\}$ is the maximum time possible between when nodes i and j send their messages.

To calculate an upper bound for $t_{send_j} - t_{send_i}$, a maximum allowed skew between local clocks at the start of the algorithm must first be defined. The constant $D_{start_{max}}$ is defined to be this limit. In other words,

$$\begin{aligned} D_{ij}(t_{send_i}) &\leq D_{start_{max}} \\ D_{ij}(t_{send_i}) &\geq -D_{start_{max}}. \end{aligned}$$

Note that a node which is $D_{start_{max}}$ behind another node at the start of the algorithm is not guaranteed to send its messages exactly $D_{start_{max}}$ time units after the first. It may send them later, due to the possibility of the node's local clock counting slower than real time.

What *is* guaranteed, however, is that if two nodes, i and j, begin with

$$\begin{aligned} D_{ij}(t_{send_i}) &= D_{start_{max}} \\ \Rightarrow \quad C_i(t_{send_i}) &= C_j(t_{send_i}) + D_{start_{max}}, \end{aligned}$$

then

$$C_j(t_{send_j}) = C_j(t_{send_i}) + D_{start_{max}}.$$

In other words, if the two nodes begin with their clocks $D_{start_{max}}$ apart, then between the times when node i and node j send their messages, node j's clock must count off exactly $D_{start_{max}}$.

This fact can be used with equation (4.2) to determine the maximum time that can elapse between when nodes i and j send their messages:

$$\begin{aligned} t_{send_j} - t_{send_i} &\leq (1+\rho)(C_j(t_{send_j}) - C_j(t_{send_i})) \\ t_{send_j} - t_{send_i} &\geq (1+\rho)^{-1}(C_j(t_{send_j}) - C_j(t_{send_i})) \end{aligned}$$

$$\Rightarrow \quad t_{send_j} - t_{send_i} \leq (1+\rho)D_{start_{max}}$$

$$t_{send_j} - t_{send_i} \geq (1+\rho)^{-1} D_{start_{max}}. \tag{4.12}$$

Finally, substituting the largest value of $t_{send_j} - t_{send_i}$ into equation (4.11) yields

$$dt_{wait_{req}} = (1+\rho) D_{start_{max}} + \ell_{max}$$

and

$$\tau_{wait_{req}} = (1+\rho)^2 D_{start_{max}} + (1+\rho)\ell_{max}.$$

Of course, if a node counts out $\tau_{wait_{req}}$ on its local clock, the maximum interval of real time $dt_{wait_{max}}$ a node may actually wait is

$$dt_{wait_{max}} = (1+\rho)^3 D_{start_{max}} + (1+\rho)^2 \ell_{max}. \tag{4.13}$$

The maximum interval $dt_{set_{max}}$ is calculated by

$$dt_{set_{max}} = \max\{t_{send_i} - t_{start}\} + dt_{wait_{max}} + dt_{calc_{max}}.$$

With the condition that $t_{start} \neq t_{send_i}$ and using equations (4.12) and (4.13), the expression becomes

$$dt_{set_{max}} = \left((1+\rho)^3 + (1+\rho)\right) D_{start_{max}} + (1+\rho)^2 \ell_{max} + dt_{calc_{max}}. \tag{4.14}$$

From here, it is possible to determine the maximum synchronization possible. Define D_{max} to be the constant which bounds clock skew at all times. Then,

$$D_{max} \geq D_{start_{max}} + dr \cdot dt_{set_{max}},$$

where $dr \cdot dt_{set_{max}}$ is the maximum drift possible between nodes from the start time of the algorithm to the time when the last node adjusts its clock.

Substituting for $dt_{set_{max}}$ via equation (4.14) gives

$$D_{max} \geq dr\left(\left((1+\rho)^3 + (1+\rho) + dr^{-1}\right) D_{start_{max}} + (1+\rho)^2 \ell_{max} + dt_{calc_{max}}\right).$$

Fortunately, a bound for $D_{start_{max}}$ has already been calculated in the guise of bounds for $D_{\mathbf{12}}(t^{(2)}_{start})$ in equation (4.10), where $D_{start_{max}} = \max\left\{\left|D_{\mathbf{12}}(t^{(2)}_{start})\right|\right\}$. Therefore,

$$\begin{aligned} D_{max} \quad \geq \quad & dr\left(\left((1+\rho)^3 + (1+\rho) + dr^{-1}\right)\right. \\ & \cdot \left((1+\rho)\ell_{max} - \left(1 + (1+\rho)^2\right)^{-1} \cdot ((1+\rho)\ell_{max} + (1+\rho)\ell_{min}) + dr \cdot dt_{between}\right) \\ & \left. + (1+\rho)^2 \ell_{max} + dt_{calc_{max}}\right). \end{aligned}$$

Determining timing constraints

The final step of analysis is to determine the minimum $dt_{between}$ that will guarantee correct operation of the algorithm over multiple rounds. There are two conditions on the minimum length of time between rounds to ensure correct operation:

$$\begin{aligned} \forall \text{ nodes } i, j, \quad t^{(2)}_{send_i} + \ell_{min} \quad & \geq \quad t_{set_j} \\ \forall \text{ nodes } i, \qquad\qquad T^{(2)}_{send} \quad & \geq \quad C_i(t_{set_i}). \end{aligned}$$

In the first condition, the time at which each node sends its messages for round two must be late enough that those messages arrive at the other nodes after those nodes have adjusted their clocks from round one. If this were not the case, a node receiving a message for round two would be timestamping using a clock that was not yet updated given the

information in the first round. The incorrect timestamp might cause the node incorrectly approximate the source node's relative clock skew.

In the second condition, for each node, after the local clock is adjusted, the local clock should read less than $T_{send}^{(2)}$, the local time at which the nodes are to send their messages for the next round of synchronization. For intuition as to why this is a requirement, consider if a node's clock were adjusted to a time beyond the point at which it was to send its next round of messages. Possibly, the node may never recognize that it didn't send its messages at the correct time, but more probably, the node would realize it was late and send the messages immediately. Unfortunately, other nodes would perceive the local clock of that node as being at an earlier time than is correct, i.e. they would not realize how ahead that node's clock actually is. Therefore, the other nodes would not adjust their clocks forward enough, and the same synchronization guarantees could not be maintained.

To begin, recall that

$$\forall \text{ nodes } i, j, \quad t_{send_i}^{(2)} + \ell_{min} \quad \geq \quad t_{set_j}.$$

For nodes **1** and **2**, this means

$$t_{send_{\mathbf{1}}}^{(2)} + \ell_{min} \quad \geq \quad t_{set_{\mathbf{2}}}$$
$$t_{send_{\mathbf{2}}}^{(2)} + \ell_{min} \quad \geq \quad t_{set_{\mathbf{1}}}.$$

Recognizing that $t_{set_i} \leq t_{send_i} + dt_{set_{max}}$ gives

$$\begin{aligned} t^{(2)}_{send_1} + \ell_{min} &\geq t_{send_2} + dt_{set_{max}} \\ t^{(2)}_{send_2} + \ell_{min} &\geq t_{send_1} + dt_{set_{max}}, \end{aligned}$$

and recalling by (4.12) that $(1+\rho)^{-1} D_{start_{max}} \leq t_{send_j} - t_{send_i} \leq (1+\rho) D_{start_{max}}$ (with $i = \mathbf{1}$, $j = \mathbf{2}$) yields

$$\begin{aligned} t^{(2)}_{send_1} + \ell_{min} &\geq t_{send_1} + (1+\rho) D_{start_{max}} + dt_{set_{max}} \\ t^{(2)}_{send_2} + \ell_{min} &\geq t_{send_2} - (1+\rho)^{-1} D_{start_{max}} + dt_{set_{max}} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad t^{(2)}_{send_1} - t_{send_1} &\geq (1+\rho) D_{start_{max}} + dt_{set_{max}} - \ell_{min} \\ t^{(2)}_{send_2} - t_{send_2} &\geq -(1+\rho)^{-1} D_{start_{max}} + dt_{set_{max}} - \ell_{min} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad dt_{between} &\geq (1+\rho) D_{start_{max}} + dt_{set_{max}} - \ell_{min} \\ dt_{between} &\geq -(1+\rho)^{-1} D_{start_{max}} + dt_{set_{max}} - \ell_{min}. \end{aligned}$$

Clearly, the first bound is more strict, so it is the only one considered now. Substituting for $dt_{set_{max}}$ via (4.14) gives

$$dt_{between} \geq (1+\rho) D_{start_{max}} + \left((1+\rho)^3 + (1+\rho) \right) D_{start_{max}} + (1+\rho)^2 \ell_{max} + dt_{calc_{max}} - \ell_{min}. \quad (4.15)$$

Substituting for $D_{start_{max}}$ gives

$$dt_{between} \geq \left((1+\rho)^3 + 2(1+\rho)\right)\left((1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}\right) + (1+\rho)^2\ell_{max} - \ell_{min} + dt_{calc_{max}}.$$

Solving for $dt_{between}$ finally gives

$$dt_{between} \geq \left[\left((1+\rho)^3 + 2(1+\rho)\right)\left((1+\rho)\ell_{max} - \ell_{assume}\right) + (1+\rho)^2\ell_{max} - \ell_{min} + dt_{calc_{max}}\right] \cdot \left(1 - dr \cdot \left((1+\rho)^3 + 2(1+\rho)\right)\right)^{-1}.$$

The second condition requires that for a node i,

$$T_{send}^{(2)} \geq C_i(t_{set_i}),$$

which can be rewritten as

$$T_{send}^{(2)} \geq T_{send} + \tau_{wait_{req}} + \tau_{calc_{max}} - \min\left\{\left[D_{ij}\right]_i\right\}$$

$$\Rightarrow \quad T_{send}^{(2)} - T_{send} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \min\left\{\left[D_{ij}\right]_i\right\}$$

$$\Rightarrow \quad \tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \min\left\{\left[D_{ij}\right]_i\right\}.$$

For nodes **1** and **2**, this becomes

$$\tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \min\{[D_{\mathbf{12}}]_{\mathbf{1}}\}$$

$$\tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \min\{[D_{21}]_2\}$$

$$\Rightarrow \tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \left((1+\rho)^{-2} D_{12}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\right)$$
$$\tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \left(\ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{12}(t_{send_1}) - \ell_{assume}\right).$$

Substituting the appropriate minimum and maximum values of $D_{12}(t_{send_1})$ gives

$$\tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume}$$
$$\tau_{between} \geq \tau_{wait_{req}} + \tau_{calc_{max}} - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{start_{max}} + \ell_{assume}.$$

Since the second lower bound is stricter, it is the only one considered further. Substituting for $\tau_{wait_{req}}$ gives

$$\tau_{between} \geq (1+\rho)^2 D_{start_{max}} + (1+\rho)\ell_{max} + \tau_{calc_{max}} - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{start_{max}} + \ell_{assume}$$

$$\Rightarrow \tau_{between} \geq \left(1 + (1+\rho)^2\right) D_{start_{max}} + (1+\rho)\ell_{max} + \tau_{calc_{max}} - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + \ell_{assume}. \quad (4.16)$$

Substituting for $D_{start_{max}}$ gives

$$\tau_{between} \geq \left(1 + (1+\rho)^2\right)((1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}) + \left(1 + \rho + \frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{min} + \ell_{assume} + \tau_{calc_{max}}.$$

The analysis has resulted in a lower bound on $\tau_{between}$ described in terms of $dt_{between}$. Since a bound for $dt_{between}$ has already been determined from the first condition, the bound for $\tau_{between}$ may be found by substitution.

4.3.2 Characterization of Algorithm for Three or More Nodes

The analysis of the failstop algorithm for three or more nodes proceeds similarly to the case of two nodes. The different results for three or more nodes stem from a removal of assumptions implicit in the two-node case. Specifically, in the two-node case, if node **2** was sufficiently behind node **1**, node **1** was guaranteed *not* to adjust its clock forward. In the case of three or more nodes, that assumption can no longer be made, since the leading node may adjust its clock based on its approximation of the clock of a node other than the trailing node. Similarly, even if the clocks of the leading and trailing nodes are within a range that permits both nodes to adjust their clocks forward, the existence of additional nodes loosens the bounds of how far forward the leading node may adjust its clock forward. The broken assumptions result in less favorable bounds on the achievable synchronization.

Define *n* nodes, **1**, **2**,...,**n**. Without loss of generality, let

$$C_{\mathbf{1}}(t_{start}) \geq C_{\mathbf{2}}(t_{start}) \geq ... \geq C_{\mathbf{n}}(t_{start}).$$

Once again, $t_{start} = t_{send_{\mathbf{1}}}$.

Determining each node's perception of local clock difference

By the same analysis performed for the two-node case, each node's perception of the others' clocks may be bounded as follows:

$\forall$ nodes $\mathbf{j} \in \mathbf{1}, ..., \mathbf{n}$:

$\forall$ nodes $\mathbf{i} < \mathbf{j}$:

$$\begin{aligned} \left[D_{\mathbf{ji}}\right]_{\mathbf{j}} &\leq (1+\rho)\ell_{max} - D_{\mathbf{ij}}(t_{send_{\mathbf{i}}}) - \ell_{assume} \\ \left[D_{\mathbf{ji}}\right]_{\mathbf{j}} &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{ij}}(t_{send_{\mathbf{i}}}) - \ell_{assume} \end{aligned}$$

$\forall$ nodes $\mathbf{k} > \mathbf{j}$:

$$\begin{aligned} \left[D_{\mathbf{jk}}\right]_{\mathbf{j}} &\leq (1+\rho)^2 D_{\mathbf{jk}}(t_{send_{\mathbf{j}}}) + (1+\rho)\ell_{max} - \ell_{assume} \\ \left[D_{\mathbf{jk}}\right]_{\mathbf{j}} &\geq (1+\rho)^{-2} D_{\mathbf{jk}}(t_{send_{\mathbf{j}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}. \end{aligned}$$

Note that for node **1**, the leading node, the first set of bounds do not apply, and for node **n**, the trailing node, the second set of bounds do not apply.

Determining limits of behavior

Using the above bounds, conditions are established for the behavior of the nodes. For each node $\mathbf{i} \in \mathbf{1}, ..., \mathbf{n}$, the following questions are considered:

1. Under what conditions is it possible for node **i** to set its clock?

2. Under what conditions is it possible for node **i** to not set its clock?

To determine the necessary conditions, recall that a node determines whether to adjust its clock based on its approximation of all clock skews. The function used by node **i** to calculate the magnitude of the adjustment may be expressed as

$$adjust_{\mathbf{i}}([D_{\mathbf{i1}}]_{\mathbf{i}}, [D_{\mathbf{i2}}]_{\mathbf{i}}, ..., [D_{\mathbf{in}}]_{\mathbf{i}}) = \begin{cases} -\min\limits_{\substack{\mathbf{j}=\mathbf{1}..\mathbf{n} \\ j \neq i}} \left\{[D_{ij}]\right\} & , \min\limits_{\substack{\mathbf{j}=\mathbf{1}..\mathbf{n} \\ j \neq i}} \left\{[D_{ij}]\right\} < 0 \\ 0 & , \min\limits_{\substack{\mathbf{j}=\mathbf{1}..\mathbf{n} \\ j \neq i}} \left\{[D_{ij}]\right\} \geq 0 \end{cases}.$$

In other words, each node selects the smallest value among its approximations of clock skews. If that value is smaller than zero, the node's clock is adjusted forward by that

amount, otherwise, the clock is not adjusted.

To find the conditions that make it possible for node **i** to adjust its clock, the conditions must be found that satisfy

$$\min\left\{\min_{\substack{\mathbf{j=1..n}\\ j\neq i}}\left\{[D_{ij}]\right\}\right\} < 0,$$

i.e., the largest possible value of $adjust_{\mathbf{i}}()$ is greater than zero. Similarly, to find the conditions that make it possible for node **i** to leave its clock alone, the conditions must be found that satisfy

$$\max\left\{\min_{\substack{\mathbf{j=1..n}\\ j\neq i}}\left\{[D_{ij}]\right\}\right\} \geq 0,$$

i.e., the smallest possible value of $adjust_{\mathbf{i}}()$ is zero.

Recall that for any node **i**:

$$D_{\mathbf{1i}}(t_{send_{\mathbf{1}}}) \geq D_{\mathbf{2i}}(t_{send_{\mathbf{1}}}) \geq \ldots \geq D_{\mathbf{ni}}(t_{send_{\mathbf{1}}})$$

$$D_{\mathbf{i1}}(t_{send_{\mathbf{1}}}) \leq D_{\mathbf{i2}}(t_{send_{\mathbf{1}}}) \leq \ldots \leq D_{\mathbf{in}}(t_{send_{\mathbf{1}}}).$$

Therefore, for node **1**,

$$\begin{aligned} & \min\left\{\min_{\mathbf{j=2..n}}\left\{[D_{\mathbf{1j}}]\right\}\right\} < 0 \\ \Rightarrow\quad & (1+\rho)^{-2}D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume} < 0 \\ \Rightarrow\quad & D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) < (1+\rho)^{2}\ell_{assume} - (1+\rho)\ell_{min}. \end{aligned}$$

Thus, the possibility of node **1** adjusting its clock is dependent on the relative skew of its closest neighbor.

In addition,

$$\begin{aligned}
\max\left\{\min_{\mathbf{j=2..n}}\left\{[D_{\mathbf{1j}}]\right\}\right\} &\geq 0 \\
\Rightarrow \quad (1+\rho)^2 D_{\mathbf{12}}(t_{send_{\mathbf{j}}}) + (1+\rho)\ell_{max} - \ell_{assume} &\geq 0 \\
\Rightarrow \quad D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) &\geq (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{max} \\
\rightarrow \quad D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) &\geq 0.
\end{aligned}$$

Again, the possibility of node **1** not adjusting its clock is dependent on the relative skew of its closest neighbor.

Continuing, for node $\mathbf{i} \in \mathbf{2}..\mathbf{n}$,

$$\begin{aligned}
\min\left\{\min_{\substack{\mathbf{j=1..n} \\ j\neq i}}\left\{[D_{\mathbf{ij}}]\right\}\right\} &< 0 \\
\Rightarrow \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1i}}(t_{send_{\mathbf{1}}}) - \ell_{assume} &< 0 \\
\Rightarrow \quad D_{\mathbf{1i}}(t_{send_{\mathbf{1}}}) &> \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} \\
\rightarrow \quad D_{\mathbf{1i}}(t_{send_{\mathbf{1}}}) &\geq 0.
\end{aligned}$$

It is always possible for these nodes to adjust their clocks forward.

Next it is determined under what conditions it is possible for nodes **2**..**n** to not adjust their clocks at all:

$$\begin{aligned}
\max\left\{\min_{\substack{\mathbf{j=1..n} \\ j\neq i}}\left\{[D_{\mathbf{ij}}]\right\}\right\} &\geq 0 \\
\Rightarrow \quad (1+\rho)\ell_{max} - D_{\mathbf{1i}}(t_{send_{\mathbf{i}}}) - \ell_{assume} &\geq 0 \\
\Rightarrow \quad D_{\mathbf{1i}}(t_{send_{\mathbf{1}}}) &\leq (1+\rho)\ell_{max} - \ell_{assume}
\end{aligned}$$

Thus the possibility for these nodes exists only when the relative skew of the leading node **1** is small enough.

Now it is possible to identify the set of initial conditions of the nodes' clocks that require independent analysis. Once again, the set of relevant initial conditions will depend on whether

$$(1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min} \stackrel{?}{>} (1+\rho)\ell_{max} - \ell_{assume}.$$

If

$$\ell_{assume} > \ell_{mid} = ((1+\rho)\ell_{max} + (1+\rho)\ell_{min})\left(1 + (1+\rho)^2\right)^{-1},$$

then the statement is true.

Thus, the list of all initial conditions that may need to be independently analyzed is as follows:

1. $\forall$ nodes $\mathbf{i} \in \mathbf{2}..\mathbf{n}, D_{\mathbf{1i}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$

2. $\forall$ nodes $\mathbf{i} \in \mathbf{2}..\mathbf{n\text{-}1}, D_{\mathbf{1i}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$

3. $\forall$ nodes $\mathbf{i} \in \mathbf{2}..\mathbf{n\text{-}1}, D_{\mathbf{1i}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

4. $\forall$ nodes $\mathbf{i} \in \mathbf{2}..\mathbf{n\text{-}2}, D_{\mathbf{1i}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $\forall$ nodes $\mathbf{i} \in \mathbf{n\text{-}1}..\mathbf{n}, D_{\mathbf{1i}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$

...

5. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $\forall$ nodes $\mathbf{i} \in \mathbf{3}..\mathbf{n}, D_{\mathbf{1i}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

6. $\forall$ nodes $\mathbf{i} \in \mathbf{2}..\mathbf{n}, D_{\mathbf{1i}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

In total, there are $\frac{1}{2}n(n+1)$ number of different ways for the nodes' clocks to reside in different regions. Fortunately, most of these configurations may be ignored.

Recall that the goal of distinguishing between initial conditions is to identify which conditions may result in different limits of synchronization after the possible adjustment of all the clocks. Keeping this in mind, it is possible to eliminate many of the possible initial conditions that result in equivalent synchronization limits.

In fact, the number of interesting cases may be reduced to the following:

1. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$

2. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$

3. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

4. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$

5. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

6. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$

For the case of $\ell_{assume} < \ell_{mid}$, the following initial condition cases are interesting:

1. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$

2. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}]$

3. $D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, \infty)$

4. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}]$

5. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, (1+\rho)\ell_{max} - \ell_{assume}]$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, \infty)$

6. $D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, \infty)$
   $D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, \infty)$

Determining limits of synchronization

The synchronization of two nodes, **i** and **j**, at the start of the next synchronization round is given by

$$\begin{aligned} D_{\mathbf{ij}}(t^{(2)}_{start}) &\leq D_{\mathbf{ij}}(t_{start}) + \Delta_{sync_{max_{\mathbf{ij}}}} + dr \cdot dt_{between} \\ D_{\mathbf{ij}}(t^{(2)}_{start}) &\geq D_{\mathbf{ij}}(t_{start}) + \Delta_{sync_{min_{\mathbf{ij}}}} - dr \cdot dt_{between}, \end{aligned}$$

where $\Delta_{sync_{min_{\mathbf{ij}}}}$ and $\Delta_{sync_{max_{\mathbf{ij}}}}$ are the minimum and maximum change in clock skew between nodes **i** and **j** due to adjustment in a single round, and $dr \cdot dt_{between}$ is the maximum drift between two local clocks possible between the start of two rounds. If $C_{\mathbf{i}}(t_{start}) > C_{\mathbf{j}}(t_{start})$, then $\Delta_{sync_{min_{\mathbf{ij}}}}$ and $\Delta_{sync_{max_{\mathbf{ij}}}}$ may be broken down further into

$$\Delta_{sync_{min_{\mathbf{ij}}}} = \min\{adjust_{\mathbf{i}}()\} - \max\left\{adjust_{\mathbf{j}}()\right\}$$
$$\Delta_{sync_{max_{\mathbf{ij}}}} = \max\{adjust_{\mathbf{i}}()\} - \min\left\{adjust_{\mathbf{j}}()\right\},$$

where $adjust_{\mathbf{i}}()$ is the amount node **i** adjusts its clock forward and $adjust_{\mathbf{j}}()$ is the amount node **j** adjusts its clock forward.

There are three interesting clock skews for which bounds will be analyzed for each of the 12 initial conditions listed above: $D_{\mathbf{12}}(t^{(2)}_{start})$, $D_{\mathbf{1n}}(t^{(2)}_{start})$, and $D_{\mathbf{2n}}(t^{(2)}_{start})$. The bounds of these three skews are guaranteed to be equal to or greater to those of any other, since $\forall$ nodes $\mathbf{i} \in \mathbf{3}..\mathbf{n\text{-}1}, C_{\mathbf{n}}(t_{send_1}) \leq C_{\mathbf{i}}(t_{send_1}) \leq C_{\mathbf{1}}(t_{send_1})$, making analysis of any other bounds unnecessary.

Case of $\ell_{assume} > \ell_{min}$:

First note the following implications:

$$D_{\mathbf{12}}(t_{send_1}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]$$

$$\begin{aligned}
\rightarrow \quad \min\{adjust_{\mathbf{1}}()\} &= 0 \\
\rightarrow \quad \max\{adjust_{\mathbf{1}}()\} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
\rightarrow \quad \min\{adjust_{\mathbf{2}}()\} &= 0 \\
\rightarrow \quad \max\{adjust_{\mathbf{2}}()\} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume}
\end{aligned}$$

$$D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}]$$

$$
\begin{aligned}
\rightarrow \quad \min\{adjust_{\mathbf{1}}()\} &= 0 \\
\rightarrow \quad \max\{adjust_{\mathbf{1}}()\} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
\rightarrow \quad \min\{adjust_{\mathbf{2}}()\} &= -(1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + \ell_{assume} \\
\rightarrow \quad \max\{adjust_{\mathbf{2}}()\} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + \ell_{assume}
\end{aligned}
$$

$$
D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)
$$

$$
\begin{aligned}
\rightarrow \quad \min\{adjust_{\mathbf{1}}()\} &= 0 \\
\rightarrow \quad \max\{adjust_{\mathbf{1}}()\} &= 0 \\
\rightarrow \quad \min\{adjust_{\mathbf{2}}()\} &= -(1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + \ell_{assume} \\
\rightarrow \quad \max\{adjust_{\mathbf{2}}()\} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + \ell_{assume}
\end{aligned}
$$

$$
D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) \in [0, (1+\rho)\ell_{max} - \ell_{assume}]
$$

$$
\begin{aligned}
\rightarrow \quad \min\{adjust_{\mathbf{n}}()\} &= 0 \\
\rightarrow \quad \max\{adjust_{\mathbf{n}}()\} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) + \ell_{assume}
\end{aligned}
$$

$$
D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) \in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
$$

$$
\begin{aligned}
\rightarrow \quad \min\{adjust_{\mathbf{n}}()\} &= -(1+\rho)\ell_{max} + D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) + \ell_{assume} \\
\rightarrow \quad \max\{adjust_{\mathbf{n}}()\} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) + \ell_{assume}
\end{aligned}
$$

Next, the minimum and maximum relative changes in clock skew between nodes **1**, **2**,

and **n** are determined for each set of initial conditions listed in the previous section. Since the results of some sets of initial conditions are the same, they have been combined below for brevity.

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\
D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}]
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{12}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{12}}}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{1n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{1n}}}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{2n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{2n}}}} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) + \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\
D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{12}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{12}}}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{1n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_{\mathbf{1}}}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{1n}}}} &= -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}
\end{aligned}
$$

$$+(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{2n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{2n}}}} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}] \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{12}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{12}}}} &= -(1+\rho)^{-2}D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{1n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{1n}}}} &= -(1+\rho)^{-2}D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{2n}}}} &= -(1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +\ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{2n}}}} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$D_{\mathbf{12}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$$

$$D_{\mathbf{1n}}(t_{send_1}) \in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{12}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{12}}}} &= (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{1n}}}} &= \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{1n}}}} &= (1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad \Delta_{sync_{min_{\mathbf{2n}}}} &= -(1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +\ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} \\
\rightarrow \quad \Delta_{sync_{max_{\mathbf{2n}}}} &= -\ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume}
\end{aligned}$$

With the maximum and minimum change in clock skew determined for all cases, the next step is plug these values into the appropriate synchronization equation.

Each resulting maximum bound on $D_{\mathbf{ij}}(t^{(2)}_{start})$ is listed below:

$$\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\
D_{\mathbf{1n}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}]
\end{aligned}$$

$$\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{start}) - (1+\rho)^{-2}D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq D_{\mathbf{1n}}(t_{start}) - (1+\rho)^{-2}D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq D_{\mathbf{2n}}(t_{start}) - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} + dr \cdot dt_{between}
\end{aligned}$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{start}) - (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq D_{\mathbf{1n}}(t_{start}) - (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq D_{\mathbf{2n}}(t_{start}) - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}] \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{start}) - (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq D_{\mathbf{1n}}(t_{start}) - (1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq D_{\mathbf{2n}}(t_{start}) - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty) \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq D_{\mathbf{12}}(t_{start}) + (1+\rho)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq D_{\mathbf{1n}}(t_{start}) + (1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq D_{\mathbf{2n}}(t_{start}) - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad +(1+\rho)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} + dr \cdot dt_{between}
\end{aligned}
$$

Noting that $D_{\mathbf{2n}}(t_{send_1}) = D_{\mathbf{1n}}(t_{send_1}) - D_{\mathbf{12}}(t_{send_1})$ and substituting the appropriate minimum or maximum values for $D_{\mathbf{12}}(t_{send_1})$ and $D_{\mathbf{1n}}(t_{send_1})$ to yield the largest bounds, the above equations become

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq \left((1+\rho) - (1+\rho)^{-1}\right)\ell_{max} + (1+\rho)^{-2}\ell_{assume} - (1+\rho)^{-1}\ell_{min} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\leq \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\leq (1+\rho)\ell_{max} - \ell_{assume} + dr \cdot dt_{between}
\end{aligned}
$$

$$\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) \quad \leq \quad \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}$$

Comparing all the possible upper bounds reveals that the third condition,

$$D_{\mathbf{2n}}(t^{(2)}_{start}) \leq \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between},$$

gives the greatest possible bound for any skew between two nodes. Note that, unlike the case of two nodes, the upper bound has no dependence on ℓ_{assume}.

The same analysis is also done for the lower bounds.

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\ D_{\mathbf{1n}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \end{aligned}$$

$$\begin{aligned} \rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\ \rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\geq D_{\mathbf{1n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\ \rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\geq D_{\mathbf{2n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \end{aligned}$$

$$\begin{aligned} D_{\mathbf{12}}(t_{send_1}) &\in [0, (1+\rho)\ell_{max} - \ell_{assume}] \\ D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty] \end{aligned}$$

$$\begin{aligned} \rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\ \rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\geq D_{\mathbf{1n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\ \rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\geq D_{\mathbf{2n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \end{aligned}$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, (1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}] \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)\ell_{max} - \ell_{assume}, \infty]
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\geq D_{\mathbf{1n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\geq D_{\mathbf{2n}}(t_{start}) - (1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
D_{\mathbf{12}}(t_{send_1}) &\in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty) \\
D_{\mathbf{1n}}(t_{send_1}) &\in ((1+\rho)^2\ell_{assume} - (1+\rho)\ell_{min}, \infty)
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\geq D_{\mathbf{12}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{12}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\geq D_{\mathbf{1n}}(t_{start}) + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\geq D_{\mathbf{2n}}(t_{start}) - (1+\rho)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume} \\
&\quad + \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - D_{\mathbf{1n}}(t_{send_1}) - \ell_{assume} - dr \cdot dt_{between}
\end{aligned}
$$

The above inequalities reduce to

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t^{(2)}_{start}) &\geq \ell_{min} - \left(1 + \rho + \frac{\rho}{1+\rho}\right)\ell_{max} - dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - dr \cdot dt_{between}
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow \quad D_{\mathbf{12}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{1n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{assume} - dr \cdot dt_{between} \\
\rightarrow \quad D_{\mathbf{2n}}(t_{start}^{(2)}) \quad &\geq \quad \ell_{min} - \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - dr \cdot dt_{between}
\end{aligned}
$$

Comparing all the above bounds reveals that the third bound,

$$
D_{\mathbf{2n}}(t_{start}^{(2)}) \geq \ell_{min} - \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - dr \cdot dt_{between},
$$

provides the smallest possible lower bound. Again, note that this lower bound is not dependent on ℓ_{assume}.

Furthermore, note that

$$\left|\min\left\{D_{\mathbf{2n}}(t^{(2)}_{start})\right\}\right| = \left|\max\left\{D_{\mathbf{2n}}(t^{(2)}_{start})\right\}\right|$$

$$\Rightarrow \left|\ell_{min} - \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - dr\cdot dt_{between}\right| = \left|\left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr\cdot dt_{between}\right|.$$

In other words, neither the maximum upper bound nor the minimum lower bound dominates the other.

So finally, for the case $\ell_{assume} > \ell_{mid}$, the limit of synchronization at the start of the next round may be expressed as

$$\forall \text{ nodes } \mathbf{i},\mathbf{j} \in \mathbf{1}..\mathbf{n},\ \left|D_{\mathbf{ij}}(t^{(2)}_{start})\right| \le \left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr\cdot dt_{between} \qquad (4.17)$$

Case of $\ell_{assume} < \ell_{min}$:

The details of the second case will be omitted, since the analysis is very similar to the case of $\ell_{assume} > \ell_{min}$. The resulting synchronization limit is also independent of ℓ_{assume} and turns out to be the same as the result above. Thus, the bounds expressed in (4.17) hold for all values of ℓ_{assume}.

Comparing the synchronization bounds for the case of three or more nodes (4.17) with that of the case of only two nodes (4.10) shows that the case of three or more nodes permits weaker synchronization.

Determining maximum synchronization

Once again, it is necessary to realize that the bounds on $\left|D_{\mathbf{ij}}(t^{(2)}_{start})\right|$ described in the pre-

vious section do not directly translate into an upper bound on clock skew for the algorithm. To determine the actual synchronization achievable, it is necessary to determine the largest clock skew possible during the execution of the algorithm, a value that was ignored in previous analysis. Although much of the analysis from the two-node case may be reused, once again the existence of additional nodes relaxes some previous assumptions, and hence, the achievable bounds.

Any increase in clock skew between two nodes during execution will be caused by two factors, clock drift and leading nodes adjusting their clock forward before the trailing nodes do. The latter reason could occur if the leading node incorrectly calculates a closely trailing node's clock as ahead of its own.

Thus, an expression for D_{max}, the constant that bounds actual maximum clock skew, may be expressed as

$$D_{max} \geq \max_{\mathbf{i,j} \in \mathbf{1..n}} \left\{ D_{\mathbf{ij}}(t_{start}) + adjust_{\mathbf{i}}() \right\} + dr \cdot dt_{set_{max}}.$$

In other words, the bound is established both by clock drift and by the maximum possible skew between any two nodes caused by adjustment of a clock.

The value of $dt_{set_{max}}$ is the same as that calculated for the two-node case in (4.14). Specifically,

$$dt_{set_{max}} = \left((1+\rho)^3 + (1+\rho) \right) D_{start_{max}} + (1+\rho)^2 \ell_{max} + dt_{calc_{max}}.$$

Additional work is required to find the maximum possible skew via clock adjustment, since this factor was not relevant in the two-node case. There are two possible worst-case contributors:

1. $D_{\mathbf{1n}}(t_{send_1}) = D_{start_{max}}$

$D_{\mathbf{12}}(t_{send_1}) \in [0, D_{start_{max}}]$

$D_{\mathbf{2n}}(t_{send_1}) = D_{\mathbf{1n}}(t_{send_1}) - D_{\mathbf{12}}(t_{send_1})$

$adjust_{\mathbf{2}}() = -\ell_{min} + \left(\frac{1}{1+\rho}\right)\ell_{max} + D_{\mathbf{12}}(t_{send_1}) + \ell_{assume}$

2. $D_{\mathbf{1n}}(t_{send_1}) = D_{start_{max}}$

   $D_{\mathbf{12}}(t_{send_1}) = 0$

   $adjust_{\mathbf{1}}() = -(1+\rho)^{-2} D_{\mathbf{12}}(t_{send_1}) - (1+\rho)^{-1}\ell_{min} + \ell_{assume}$

For the first case above,

$$\begin{aligned} D_{max} &\geq \max\{D_{\mathbf{2n}}(t_{start}) + adjust_{\mathbf{2}}()\} + dr \cdot dt_{set_{max}} \\ \Rightarrow \quad D_{max} &\geq D_{start_{max}} - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + \ell_{assume} + dr \cdot dt_{set_{max}} \end{aligned}$$

For the second case,

$$\begin{aligned} D_{max} &\geq \max\{D_{\mathbf{1n}}(t_{start}) + adjust_{\mathbf{1}}()\} + dr \cdot dt_{set_{max}} \\ \Rightarrow \quad D_{max} &\geq D_{start_{max}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume} + dr \cdot dt_{set_{max}} \end{aligned}$$

Comparing the two results shows that the first bound is strictly larger than the second, and is thus the one of interest. Furthermore, note that by choosing $\ell_{assume} = \ell_{min}$, the bound may be minimized. Thus, the achievable synchronization is

$$D_{max} \geq D_{start_{max}} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + dr \cdot dt_{set_{max}}.$$

By substituting equation (4.14) for $dt_{set_{max}}$, the following expression can be obtained:

$$D_{max} \geq \left(1 + dr\left((1+\rho)^3 + (1+\rho)\right)\right) D_{start_{max}} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + dr \cdot (1+\rho)^2 \ell_{max} + dr \cdot dt_{calc_{max}}.$$

Substituting $D_{start_{max}} = \max\left\{\left|D_{\mathbf{ij}}(t^{(2)}_{start})\right|\right\}$ from (4.17) yields

$$\begin{aligned} D_{max} \quad \geq \quad & \left(1 + dr\left((1+\rho)^3 + (1+\rho)\right)\right) \cdot \left(\left(1 + \rho + \left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}\right) \\ & + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + dr \cdot (1+\rho)^2 \ell_{max} + dr \cdot dt_{calc_{max}}. \end{aligned} \tag{4.18}$$

Determining timing constraints

As in the two-node case, the minimum time between synchronization rounds required for correct execution is constrained by

$$\begin{aligned} \forall \text{ nodes } i, j, \quad t^{(2)}_{send_i} + \ell_{min} \quad & \geq \quad t_{set_j} \\ \forall \text{ nodes } i, \qquad\qquad T^{(2)}_{send} \quad & \geq \quad C_i(t_{set_i}). \end{aligned}$$

The first condition requires that no message sent in the next synchronization round should reach a node before the node has completed adjusting its clock in the current round. The second condition requires that no node must ever adjust its clock forward beyond the next time at which it needs to send the next round of messages.

To satisfy the first condition for the case of three or more nodes, it is possible to reuse (4.15), since the analysis is the same up to that point. The expression is reprinted here for convenience:

$$dt_{between} \geq (1+\rho)D_{start_{max}} + \left((1+\rho)^3 + (1+\rho)\right) D_{start_{max}} + (1+\rho)^2 \ell_{max} + dt_{calc_{max}} - \ell_{min}.$$

Substituting the new value for $D_{start_{max}}$ calculated for the current case of three or more

nodes gives

$$dt_{between} \geq \left((1+\rho)^3 + 2(1+\rho)\right) \cdot \left(\left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}\right) + (1+\rho)^2\ell_{max} + dt_{calc_{max}} - \ell_{min}.$$

Solving for $dt_{between}$ finally gives

$$dt_{between} \geq \left[\left((1+\rho)^3 + 2(1+\rho)\right)\left(\left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min}\right) + (1+\rho)^2\ell_{max} - \ell_{min} + dt_{calc_{max}}\right] \cdot \left(1 - dr \cdot \left((1+\rho)^3 + 2(1+\rho)\right)\right)^{-1}. \quad (4.19)$$

To satisfy the second condition, it is possible to reuse (4.16), since the analysis is the same up to that point. The expression is reprinted here for convenience:

$$\tau_{between} \geq \left(1 + (1+\rho)^2\right) D_{start_{max}} + (1+\rho)\ell_{max} + \tau_{calc_{max}} - \ell_{min} + \left(\frac{\rho}{1+\rho}\right)\ell_{max} + \ell_{assume}.$$

Substituting for $D_{start_{max}}$ gives

$$\tau_{between} \geq \left(1 + (1+\rho)^2\right)\left(\left(1+\rho+\left(\frac{\rho}{1+\rho}\right)\right)\ell_{max} - \ell_{min} + dr \cdot dt_{between}\right) + \left(1+\rho+\frac{\rho}{1+\rho}\right)\ell_{max} - \ell_{min} + \ell_{assume} + \tau_{calc_{max}}.$$

Once again, the analysis has resulted in a lower bound on $\tau_{between}$ described in terms of

$dt_{between}$. Since a bound for $dt_{between}$ has already been determined from the first condition, the bound for $\tau_{between}$ may be found by substitution.

4.4 Byzantine Resilient Algorithm

A Byzantine resilient algorithm is one that is executed by a group of nodes and can tolerate arbitrary incorrect actions by at least one member of the group. Algorithm 4.3.1 is not Byzantine resilient. If a single node behaved incorrectly, such as sending messages at different times to different nodes, different nodes may adjust their clocks according to inconsistent information, and the correctness of the algorithm could no longer be preserved, and no synchronization bounds could be guaranteed.

Description of Adversary

The adversary that the Byzantine resilient algorithm must cope with has the power to control the contents of whatever messages it sends, and it may also control when it sends its messages to each other node, if it sends one at all. Since connections between nodes are point-to-point, a message sent by the adversary to one node can never be seen by another node, allowing the adversary more secrecy than if there were a common broadcast medium. However, the point-to-point connection also prevents the adversary from pretending to be another node. A receiving node always knows the source of a message because there is a unique physical port associated with every other node. Only one adversary may exist in the system at any time.

Description of Algorithm

The Byzantine resilient algorithm requires three communication rounds. The first communication round requires no algorithm-specific data. It only requires that a message be sent. From the arrival times of the first messages, each node infers the others' relative clock skew. The next two rounds involve each node broadcasting its calculated clock skew data to all other nodes in a Byzantine resilient manner. The arrival times of the round 2 and round 3 messages are not considered. Thus, after the third communication round, all honest nodes are guaranteed to know the same information, i.e., every node's reported clock skew from every other node. Of course, a faulty node could report incorrect information, but every other node will know the exact same incorrect information, allowing consistent behavior among honest nodes. Once the nodes all have consistent information, they may then analyze the data, recognize faults, and adjust their clocks forward appropriately.

Algorithm 4.4.1 augments algorithm 4.3.1 with Byzantine resilient properties. The algorithm is split across pages in figures 4-3 and 4-4.

Algorithm 4.4.1: BYZ-SYNC()

Let $N = \{n_1 ... n_k\}$ be a set of k participating nodes.
Let $i \in \{1...k\}$ be the number of the node executing the instance of the algorithm.

global $timestamps[k] \leftarrow \{null, ..., null\}$
global $deltas[k][k][k] \leftarrow \{\{0, ...0\}, ..., \{0, ...0\}\}$

process RECEIVE-MESSAGES()
while $C_i(t) < T_{send} + \tau_{wait}$
do { **if** m_{ji} received
then $timestamps[j]$ = CURRENT-TIME()

while $C(t) < T'_{send} + \tau'_{wait}$
do { **if** m'_{ji} received
then { $deltas_j = m'_{ji}$
$deltas[i][j] = deltas_j$

while $C(t) < T''_{send} + \tau''_{wait}$
do { **if** m'_{ji} received
then { $deltas_j = m'_{ji}$
$deltas[j] = deltas_j$

procedure CALCULATE-DELTA($timestamp$)
$delta \leftarrow timestamp - (T_{send} + \ell_{assume})$
return ($delta$)

Figure 4-3: Part 1 of the Byzantine synchronization algorithm. Process Receive-Messages timestamps messages that arrive in the first round, and stores information for node's reported clock skews that arrive in the second and third rounds.

process SEND-AND-ADJUST()

if $C_i(t) = T_{send}$

then
- **for** $j \leftarrow 1$ **to** k, $j \neq i$
  - **do** {Send any message m_{ij} to n_j.
- **while** $C(t) \neq T_{send} + \tau_{wait}$
  - **do** { Nothing.
- **for** $j \leftarrow 1$ **to** k
  - **do**
    - **if** $j \neq i$
      - **then** $deltas[i][i][j] \leftarrow$ CALCULATE-DELTA($timestamps[j]$)

if $C_i(t) = T'_{send}$

then
- **for** $j \leftarrow 1$ **to** k, $j \neq i$
  - **do** {Send $m'_{ij} = deltas[i][i]$ to n_j.

if $C_i(t) = T''_{send}$

then
- **for** $j \leftarrow 1$ **to** k, $j \neq i$
  - **do** {Send $m''_{ij} = deltas[i]$ to n_j.
- **while** $C(t) \neq T''_{send} + \tau''_{wait}$
  - **do** { Nothing.

RESOLVE-INCONSISTENCIES($deltas$)

SET-TIME(CURRENT-TIME() − MIN($deltas$))

Figure 4-4: Part 2 of the Byzantine synchronization algorithm. Process Send-And-Adjust sends out appropriate information in each round. At the end of round three, after all nodes have consistent information, procedure Resolve-Inconsistencies uses consistency checks, described later, to recover from a possible faulty node. The clock is then adjusted as necessary.

Fault Scenarios

This section describes the various ways an adversary may affect the system. Some fault scenarios allow the honest nodes to identify the source of the failure. Others scenarios make it impossible to pinpoint the source. The scenarios described below assume a total of four nodes, three honest nodes and a single adversary. Note that the following scenarios do not include an adversary deviating from the Byzantine exchange protocol in communication rounds 2 and 3.

1. **The adversary sends no round 1 messages.** No honest node receives any messages from the faulty node. This faulty node is easily identified and may be safely ignored.

2. **The adversary sends a round 1 message to only one other node.** Here, two honest nodes believe the adversary to be faulty, and one honest node does not. Since the naive node knows that two nodes cannot falsely accuse another node (due to the assumption of one faulty node), it accepts that the adversary is faulty.

3. **The adversary sends a round 1 message to only two nodes.** In this case only one honest node knows that the adversary is faulty, and the other two are naive. However, the lone informed node cannot convince the others the adversary is faulty. The naive nodes have no way to discern if the informed node is falsely accusing the adversary, and therefore must proceed as if the adversary were behaving correctly. If the node that detected the fault is to remain correctly synchronized, it must also proceed as if the adversary is behaving correctly, but infer the adversary's clock skew via the other nodes' approximations.

Consider what would happen if the node that claims to detect the fault were trusted unconditionally. If that node were actually lying and the node under scrutiny were actually honest and ahead of the other nodes, synchronization would never be achieved. The two

naive, honest nodes would accept the lying node's claim that the leading node was faulty, and they would not adjust their clocks forward to meet the leading node. Since the leading node may not adjust its clock backward, no synchronization could be maintained.

4. The adversary sends a round 1 message to all nodes simultaneously, but at a time such that the $D_{start_{max}}$ assumption is violated for all three honest nodes. For three honest nodes **1**, **2** and **3** and the adversary **4**,

$$\begin{aligned} |[D_{\mathbf{14}}]_{\mathbf{1}}| &> D_{start_{max}} + \epsilon \\ |[D_{\mathbf{24}}]_{\mathbf{2}}| &> D_{start_{max}} + \epsilon \\ |[D_{\mathbf{34}}]_{\mathbf{3}}| &> D_{start_{max}} + \epsilon. \end{aligned}$$

The adversary is easily identified and ignored, since all three honest nodes agree that **4** is in violation of the synchronization assumption.

5. The adversary sends a round 1 message to all other nodes at the same time, but at a time such that the $D_{start_{max}}$ assumption is violated for two of the three honest nodes. In this case,

$$\begin{aligned} |[D_{\mathbf{14}}]_{\mathbf{1}}| &> D_{start_{max}} + \epsilon \\ |[D_{\mathbf{24}}]_{\mathbf{2}}| &> D_{start_{max}} + \epsilon \\ |[D_{\mathbf{34}}]_{\mathbf{3}}| &<= D_{start_{max}} + \epsilon. \end{aligned}$$

The adversary can be identified, since **1** and **2** both agree that **4** is faulty. Since the naive

node **3** knows that **1** and **2** could not both falsely implicate **4**, it accepts that **4** is faulty.

6. The adversary sends a round 1 message to all other nodes at the same time, but at a time such that the $D_{start_{max}}$ assumption is violated for only one of the honest nodes. For the three honest nodes **1**, **2**, **3** and the adversary **4**,

$$\begin{aligned} |[D_{\mathbf{14}}]_{\mathbf{1}}| &> D_{start_{max}} + \epsilon \\ |[D_{\mathbf{24}}]_{\mathbf{2}}| &<= D_{start_{max}} + \epsilon \\ |[D_{\mathbf{34}}]_{\mathbf{3}}| &<= D_{start_{max}} + \epsilon. \end{aligned}$$

In this case, the adversary's fault has only been noticed by a single honest node. From the point of view of the two naive nodes, **2** and **3**, either **4** could have sent **1** a message at an inappropriate time or **1** is misrepresenting when **4** actually sent its message to **1**. Consider the worst case,

$$\begin{aligned} [D_{\mathbf{14}}]_{\mathbf{1}} &= -2 * (D_{start_{max}} + \epsilon) \\ [D_{\mathbf{24}}]_{\mathbf{2}} &= -D_{start_{max}} + \epsilon \\ [D_{\mathbf{34}}]_{\mathbf{3}} &= -D_{start_{max}} + \epsilon. \end{aligned}$$

In other words, nodes **2** and **3**'s clocks are roughly the same, **1**'s is as far behind as possible, and **4**, the adversary, sent its messages such that **2** and **3** believe it is as far ahead as possible (and they are as far behind as possible) and **1** knows **4** is faulty. Nodes **2** and **3** can't know for sure whether their own clocks ran fast or slow since the last synchronization, so they cannot know whether it is correct for **4** to be in front. Therefore, nodes **2** and **3**

must accept **4**'s claim that it is ahead and set their clock forward appropriately, and node **1**, knowing the action that **2** and **3** will take, must accordingly set its clock drastically ahead.

Note that if **4** were incorrectly assumed to be faulty, **2** and **3** would not set their clocks forward, and the clock skew between those nodes and **4** would never be closed (since **4** may not set its clock backward), and no synchronization guarantees could be made.

7. The adversary sends a round 1 message to each other node at very different times. Honest nodes may determine whether a participant node sent all its round 1 messages at the same time. Each node may confirm its own approximation of another node's local clock by adding its approximation of a second node with that second node's approximation of the node in question. Let **1**, **2**, **3** and **4** be participating nodes. Node **1** may confirm its approximation of $[D_{12}]_1$ by checking that

$$\begin{aligned} [D_{12}]_1 &\approx [D_{13}]_1 + [D_{32}]_3 \\ [D_{12}]_1 &\approx [D_{14}]_1 + [D_{42}]_4 \,. \end{aligned}$$

Of course, the three different approximations will not be exact, due to varying message latencies and processing times, but the checks do limit the flexibility of an adversary to influence the behavior of the other nodes.

If both of those consistency checks fail, all honest nodes realize that **2** must be the culprit. Under the assumption that only one node may be faulty, the two nodes used to confirm the skew approximation cannot both be misreporting their approximation of the first node's skew (or misrepresenting their own clock). Thus, the honest nodes may agree that the adversary is faulty.

8. The adversary sends a round 1 message to two of the honest nodes at the same time, and sends to the third honest node at a different time. As in the previous case, the nodes will notice a fault because one of the consistency checks will fail. However, since only a single check fails, the nodes will not be able to pinpoint the adversary. Thus, the node that received faulty data about the adversary's clock must instead adjust its clock according an indirect approximation of the adversary. For example, if an adversary **4** gave consistent clock information to **2** and **3**, but revealed a fault to node **1**, node **1** would be forced to calculate **4**'s advertised clock skew by

$$[D_{14}]_{1(2)} \approx [D_{12}]_1 + [D_{24}]_2 \,.$$

9. The adversary misrepresents its relative clock skew with two or more nodes by broadcasting incorrect values in round 2. All nodes behave honestly in round 1, but the adversary lies in round 2, indicating that messages from two or more nodes were received by him at different times than they actually were. The honest nodes can identify this adversary because it affects two pieces of information that are independently verifiable.

Say nodes **1**, **2**, and **3** are honest, while **4** has reported incorrect approximations of $[D_{42}]_4$ and $[D_{43}]_4$. Since **1**, **2**, and **3** are honest, they report relative skews of the other nodes such that

$$\begin{aligned}
[D_{21}]_2 &\approx [D_{21}]_{2(3)} = [D_{23}]_2 + [D_{31}]_3 \\
[D_{21}]_2 &\approx [D_{21}]_{2(4)} = [D_{24}]_2 + [D_{41}]_4 \\
[D_{31}]_3 &\approx [D_{31}]_{3(2)} = [D_{32}]_3 + [D_{21}]_2 \\
[D_{31}]_3 &\approx [D_{31}]_{3(4)} = [D_{34}]_3 + [D_{41}]_4
\end{aligned}$$

$$[D_{12}]_1 \approx [D_{12}]_{1(3)} = [D_{13}]_1 + [D_{32}]_3$$

$$[D_{32}]_3 \approx [D_{32}]_{3(1)} = [D_{31}]_3 + [D_{12}]_1$$

$$[D_{13}]_1 \approx [D_{13}]_{1(2)} = [D_{12}]_1 + [D_{23}]_2$$
$$[D_{23}]_2 \approx [D_{23}]_{2(1)} = [D_{21}]_2 + [D_{13}]_1 .$$

Since **4** has been dishonest, the following approximations are not true, indicating a fault:

$$[D_{12}]_1 \not\approx [D_{12}]_{1(4)} = [D_{14}]_1 + [D_{42}]_4$$
$$[D_{32}]_3 \not\approx [D_{32}]_{3(4)} = [D_{34}]_3 + [D_{42}]_4$$

$$[D_{13}]_1 \not\approx [D_{13}]_{1(4)} = [D_{14}]_1 + [D_{43}]_4$$
$$[D_{23}]_2 \not\approx [D_{23}]_{2(4)} = [D_{24}]_2 + [D_{43}]_4$$

$$[D_{24}]_2 \not\approx -[D_{42}]_4$$
$$[D_{34}]_3 \not\approx -[D_{43}]_4 .$$

Nodes **1**, **2**, and **3** may all determine that node **4** is faulty. Nodes **2** and **3** are immediately convinced of **4**'s fault from $[D_{24}]_2 \not\approx [D_{42}]_4$ and $[D_{34}]_3 \not\approx [D_{43}]_4$, respectively. Node **1** must combine two inconsistencies to be convinced of **4**'s fault. For example, from $[D_{24}]_2 \not\approx [D_{42}]_4$, node **1** knows that either **2** or **4** is faulty. From $[D_{34}]_3 \not\approx [D_{43}]_4$, **1** knows that either **3** or **4** is faulty. By intersecting the sets of possibly faulty nodes from both, it may conclude that **4** must be the faulty node.

10. The adversary misrepresents its relative clock skew with a single other node by broadcasting a single incorrect value in round 2. All nodes behave honestly in round 1, but the adversary lies in round 2, indicating that a single message from another node was received by him at a different time than it actually was. In this case, it is not possible for the honest nodes to agree on which node is faulty.

Say nodes **1**, **2**, and **3** are honest, while **4** has reported an incorrect approximation of $[D_{43}]_4$. As a result, the following consistency checks fail:

$$[D_{13}]_1 \not\approx [D_{13}]_{1(4)} = [D_{14}]_1 + [D_{43}]_4$$
$$[D_{23}]_2 \not\approx [D_{23}]_{2(4)} = [D_{24}]_2 + [D_{43}]_4$$

$$[D_{34}]_3 \not\approx [D_{43}]_4 .$$

Both node **1** and node **2** suspect either **3** or **4** as faulty, and node **3** knows for sure that **4** is faulty. However, **3**'s certainty is of no help to **1** and **2**, since they already view **3** as suspicious. Thus, the honest nodes have no choice but to treat all other nodes as if they could be honest and adjust their clocks forward as necessary.

11. A single node commits errors in multiple communication stages. If a single node makes errors in multiple communication stages, and if it commits a grievous enough fault at any single stage to be pinpointed as faulty, its actions from other stages may be safely ignored.

Also, through a series of several faults in different stages, each honest node may be able to pinpoint the source of the two faults. For example, say nodes **1**, **2**, and **3** are honest, while **4** sent an early message to **2** in round 1 and misrepresented its skew with node **3** in

the exchange steps. Then, $[D_{24}]_2$ and $[D_{43}]_4$ are inconsistent with the rest of the data, and the following checks would not hold:

$$[D_{13}]_1 \not\approx [D_{13}]_{1(4)} = [D_{14}]_1 + [D_{43}]_4$$
$$[D_{23}]_2 \not\approx [D_{23}]_{2(4)} = [D_{24}]_2 + [D_{43}]_4$$

$$[D_{14}]_1 \not\approx [D_{14}]_{1(2)} = [D_{12}]_1 + [D_{24}]_2$$
$$[D_{34}]_3 \not\approx [D_{34}]_{3(2)} = [D_{32}]_3 + [D_{24}]_2$$

$$[D_{24}]_2 \not\approx -[D_{42}]_4$$
$$[D_{34}]_3 \not\approx -[D_{43}]_4 .$$

Nodes **1**, **2** and **3** may all determine **4** as faulty, since **4** is independently involved with multiple inconsistencies.

Determining Consistency Checks

The previous descriptions of fault scenarios discussed how nodes might discover various inconsistencies in skew data. The following sections characterize the consistency checks the nodes may use to confirm proper operation of other nodes.

Pair Consistency

The pair consistency check involves verifying whether two nodes report approximately the same relative clock skew from each other. Recall that the consistency check can only

be approximate due to unpredictable message latencies, varying processing times, and continual clock drift. Therefore, certain ranges of approximations indicate correct behavior by the other nodes, while approximations outside those ranges indicate a fault.

Consider an honest node **2** that sends a round 1 message to node **1**. Node **1** can approximate $[D_{\mathbf{12}}]_{\mathbf{1}}$ to within

$$\begin{aligned} [D_{\mathbf{12}}]_{\mathbf{1}} &\leq D_{\mathbf{12}}(t_{send_2}) + (1+\rho)\ell_{max} - \ell_{assume} \\ [D_{\mathbf{12}}]_{\mathbf{1}} &\geq D_{\mathbf{12}}(t_{send_2}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}. \end{aligned}$$

After communication rounds 2 and 3, node **1** knows **2**'s announced value of $[D_{\mathbf{21}}]_{\mathbf{2}}$ and may perform this consistency check:

$$[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}} \approx 0.$$

Since **1** knows that a correct $[D_{\mathbf{21}}]_{\mathbf{2}}$ must be bounded by

$$\begin{aligned} [D_{\mathbf{21}}]_{\mathbf{2}} &\leq D_{\mathbf{21}}(t_{send_1}) + (1+\rho)\ell_{max} - \ell_{assume} \\ [D_{\mathbf{21}}]_{\mathbf{2}} &\geq D_{\mathbf{21}}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}, \end{aligned}$$

then the acceptable error for $[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}}$ is

$$
\begin{aligned}
[D_{12}]_1 + [D_{21}]_2 &\leq D_{12}(t_{send_2}) + D_{21}(t_{send_1}) + 2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{12}]_1 + [D_{21}]_2 &\geq D_{12}(t_{send_2}) + D_{21}(t_{send_1}) + 2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
[D_{12}]_1 + [D_{21}]_2 &\leq D_{12}(t_{send_2}) - D_{12}(t_{send_1}) + 2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{12}]_1 + [D_{21}]_2 &\geq D_{12}(t_{send_2}) - D_{12}(t_{send_1}) + 2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
[D_{12}]_1 + [D_{21}]_2 &\leq dr\left|t_{send_2} - t_{send_1}\right| + 2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{12}]_1 + [D_{21}]_2 &\geq -dr\left|t_{send_2} - t_{send_1}\right| + 2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}.
\end{aligned}
$$

Loose bounds on $\left|t_{send_2} - t_{send_1}\right|$ can be provided by

$$
\begin{aligned}
\left|t_{send_2} - t_{send_1}\right| &\leq (1+\rho)D_{start_{max}} \\
\left|t_{send_2} - t_{send_1}\right| &\leq (1+\rho)^{-1}D_{start_{max}}.
\end{aligned}
$$

but stricter bounds on $\left|t_{send_2} - t_{send_1}\right|$ are possible by inferring them from the information provided by either $[D_{12}]_1$ or $[D_{21}]_2$ and using it to calculate a smaller acceptable range.

Given an approximation $[D_{12}]_1$, a node knows that the possible values of $D_{12}(t_{send_2})$ which produced it are bounded in terms of the $[D_{12}]_1$ by

$$\begin{aligned}[D_{12}]_1 &\leq D_{12}(t_{send_2}) + (1+\rho)\ell_{max} - \ell_{assume} \\ [D_{12}]_1 &\geq D_{12}(t_{send_2}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\end{aligned}$$

$$\begin{aligned}D_{12}(t_{send_2}) &\leq [D_{12}]_1 - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\ D_{12}(t_{send_2}) &\geq [D_{12}]_1 - (1+\rho)\ell_{max} + \ell_{assume}.\end{aligned}$$

A node also knows that bounds on $\left|t_{send_2} - t_{send_1}\right|$ can be provided by

$$\begin{aligned}\left|t_{send_2} - t_{send_1}\right| &\leq (1+\rho)\left|D_{12}(t_{send_2})\right| \\ \left|t_{send_2} - t_{send_1}\right| &\geq (1+\rho)^{-1}\left|D_{12}(t_{send_2})\right|.\end{aligned}$$

and the following can be obtained:

$$\begin{aligned}\left|t_{send_2} - t_{send_1}\right| &\leq (1+\rho)\max\left\{\left|[D_{12}]_1 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{12}]_1 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\ \left|t_{send_2} - t_{send_1}\right| &\geq (1+\rho)^{-1}\min\left\{\left|[D_{12}]_1 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{12}]_1 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\}.\end{aligned}$$

Finally, bounds on $[D_{12}]_1 + [D_{21}]_2$ may be obtained by substituting the above:

$$
\begin{aligned}
[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}} &\leq dr(1+\rho)\max\left\{\left|[D_{\mathbf{12}}]_{\mathbf{1}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{\mathbf{12}}]_{\mathbf{1}} - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}} &\geq -dr(1+\rho)^{-1}\min\left\{\left|[D_{\mathbf{12}}]_{\mathbf{1}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{\mathbf{12}}]_{\mathbf{1}} - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}.
\end{aligned}
$$

Thus, if the sum $[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}}$ is outside the boundaries above, that indicates a fault has taken place.

A node may similarly use $[D_{\mathbf{21}}]_{\mathbf{2}}$ in addition to $[D_{\mathbf{12}}]_{\mathbf{1}}$ by checking that

$$
\begin{aligned}
[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}} &\leq dr(1+\rho)\max\left\{\left|[D_{\mathbf{21}}]_{\mathbf{2}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{\mathbf{21}}]_{\mathbf{2}} - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{21}}]_{\mathbf{2}} &\geq -dr(1+\rho)^{-1}\min\left\{\left|[D_{\mathbf{21}}]_{\mathbf{2}} - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{\mathbf{21}}]_{\mathbf{2}} - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}.
\end{aligned}
$$

Maximum Skew Consistency

A central tenet of the algorithms described in this thesis is that the maximum clock skew between any two nodes at the start of the algorithm must be equal to or less than a quantity $D_{start_{max}}$, the value of which depends on the physical properties of the system. Any node that does not maintain this property must be faulty.

Consider two honest nodes **1** and **2** that begin the execution of the algorithm with

$$D_{\mathbf{12}}(t_{send_{\mathbf{1}}}) = D_{start_{max}}.$$

Then, since the bounds of node **2**'s skew approximation of **1** is bounded by

$$[D_{\mathbf{21}}]_{\mathbf{2}} \quad \geq \quad D_{\mathbf{21}}(t_{send_1}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume},$$

then the smallest value of $[D_{\mathbf{21}}]_{\mathbf{2}}$ that node **2** should ever report is

$$[D_{\mathbf{21}}]_{\mathbf{2}} \quad \geq \quad -D_{start_{max}} + (1+\rho)^{-1}\ell_{min} - \ell_{assume}.$$

Also, since

$$[D_{\mathbf{12}}]_{\mathbf{2}} \quad \leq \quad D_{\mathbf{12}}(t_{send_2}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume},$$

and

$$D_{\mathbf{12}}(t_{send_2}) \leq D_{\mathbf{12}}(t_{send_1}) + dr(t_{send_2} - t_{send_1}),$$

and

$$t_{send_2} - t_{send_1} \leq (1+\rho)D_{\mathbf{12}}(t_{send_1}),$$

then the largest value of $[D_{\mathbf{12}}]_{\mathbf{1}}$ that node **1** should ever report is

$$[D_{\mathbf{12}}]_{\mathbf{1}} \quad \leq \quad D_{start_{max}} + dr(1+\rho)D_{start_{max}} + (1+\rho)\ell_{max} - \ell_{assume}.$$

These bounds can generalize for the reported skew for any pair of nodes. For any two nodes **i**and **j**, a reported skew $\left[D_{\mathbf{ij}}\right]_{\mathbf{i}}$ must be within

$$\begin{aligned}\left[D_{\mathbf{ij}}\right]_{\mathbf{i}} &\leq D_{start_{max}} + dr(1+\rho)D_{start_{max}} + (1+\rho)\ell_{max} - \ell_{assume} \\ \left[D_{\mathbf{ij}}\right]_{\mathbf{i}} &\geq -D_{start_{max}} + (1+\rho)^{-1}\ell_{min} - \ell_{assume}.\end{aligned}$$

If node **i** reports otherwise, either node **i** or **j** must be faulty. Which one is faulty, however, may be impossible to determine unless one of the nodes can be implicated by an additional inconsistency.

Transitive Consistency

Transitive consistency involves verifying a node's reported relative clock skew with a second node by using reported clock skews between those nodes and a third, independent node. Again, this consistency check can only be approximate.

Consider honest nodes **1**, **2** and **3** exchanging messages. The reported skew $[D_{\mathbf{13}}]_{\mathbf{1}}$ may be verified by checking

$$\begin{aligned}[D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{23}}]_{\mathbf{2}} &\approx [D_{\mathbf{13}}]_{\mathbf{1}} \\ \Rightarrow [D_{\mathbf{12}}]_{\mathbf{1}} + [D_{\mathbf{23}}]_{\mathbf{2}} - [D_{\mathbf{13}}]_{\mathbf{1}} &\approx 0.\end{aligned}$$

The following bounds hold true for a system without faults:

$$\begin{aligned}[D_{\mathbf{13}}]_{\mathbf{1}} &\leq D_{\mathbf{13}}(t_{send_3}) + (1+\rho)\ell_{max} - \ell_{assume} \\ [D_{\mathbf{13}}]_{\mathbf{1}} &\geq D_{\mathbf{13}}(t_{send_3}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}\end{aligned}$$

$$
\begin{aligned}
[D_{12}]_1 &\leq D_{12}(t_{send_2}) + (1+\rho)\ell_{max} - \ell_{assume} \\
[D_{12}]_1 &\geq D_{12}(t_{send_2}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
[D_{23}]_2 &\leq D_{23}(t_{send_3}) + (1+\rho)\ell_{max} - \ell_{assume} \\
[D_{23}]_2 &\geq D_{23}(t_{send_3}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}.
\end{aligned}
$$

Therefore, the results of the consistency check should be bounded by

$$
\begin{aligned}
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\leq D_{12}(t_{send_2}) + D_{23}(t_{send_3}) - D_{13}(t_{send_3}) \\
&\quad +2(1+\rho)\ell_{max} - (1+\rho)^{-1}\ell_{min} - \ell_{assume} \\
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\geq D_{12}(t_{send_2}) + D_{23}(t_{send_3}) - D_{13}(t_{send_3}) \\
&\quad +2(1+\rho)^{-1}\ell_{min} - (1+\rho)\ell_{max} - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\leq D_{12}(t_{send_2}) - D_{12}(t_{send_3}) + 2(1+\rho)\ell_{max} - (1+\rho)^{-1}\ell_{min} - \ell_{assume} \\
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\geq D_{12}(t_{send_2}) - D_{12}(t_{send_3}) + 2(1+\rho)^{-1}\ell_{min} - (1+\rho)\ell_{max} - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\leq dr\left|t_{send_2} - t_{send_3}\right| + 2(1+\rho)\ell_{max} - (1+\rho)^{-1}\ell_{min} - \ell_{assume} \\
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 &\geq -dr\left|t_{send_2} - t_{send_3}\right| + 2(1+\rho)^{-1}\ell_{min} - (1+\rho)\ell_{max} - \ell_{assume}.
\end{aligned}
$$

To find the bounds of $\left|t_{send_2} - t_{send_3}\right|$, note that

$$
\begin{aligned}
[D_{23}]_2 &\leq D_{23}(t_{send_3}) + (1+\rho)\ell_{max} - \ell_{assume} \\
[D_{23}]_1 &\geq D_{23}(t_{send_3}) + (1+\rho)^{-1}\ell_{min} - \ell_{assume}
\end{aligned}
$$

$$
\begin{aligned}
D_{23}(t_{send_3}) &\leq [D_{23}]_2 - (1+\rho)^{-1}\ell_{min} + \ell_{assume} \\
D_{23}(t_{send_3}) &\geq [D_{23}]_2 - (1+\rho)\ell_{max} + \ell_{assume}.
\end{aligned}
$$

Also note that

$$
\begin{aligned}
\left|t_{send_2} - t_{send_3}\right| &\leq (1+\rho)\left|D_{23}(t_{send_3})\right| \\
\left|t_{send_2} - t_{send_3}\right| &\geq (1+\rho)^{-1}\left|D_{23}(t_{send_3})\right|,
\end{aligned}
$$

and the following can be obtained by substitution:

$$
\begin{aligned}
\left|t_{send_2} - t_{send_3}\right| &\leq (1+\rho)\max\left\{\left|[D_{23}]_2 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{23}]_2 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
\left|t_{send_2} - t_{send_3}\right| &\geq (1+\rho)^{-1}\min\left\{\left|[D_{23}]_2 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{23}]_2 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\}.
\end{aligned}
$$

Finally, bounds on $[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1$ may be obtained by substituting the above:

$$
\begin{aligned}
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 \quad &\leq \quad dr(1+\rho) \\
&\quad \cdot \max\left\{\left|[D_{23}]_2 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{23}]_2 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)\ell_{max} - 2\ell_{assume} \\
[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1 \quad &\geq \quad -dr(1+\rho)^{-1} \\
&\quad \cdot \min\left\{\left|[D_{23}]_2 - (1+\rho)^{-1}\ell_{min} + \ell_{assume}\right|, \left|[D_{23}]_2 - (1+\rho)\ell_{max} + \ell_{assume}\right|\right\} \\
&\quad +2(1+\rho)^{-1}\ell_{min} - 2\ell_{assume}.
\end{aligned}
$$

Thus, if the sum $[D_{12}]_1 + [D_{23}]_2 - [D_{13}]_1$ is outside the boundaries above, that indicates a fault has taken place.

Analysis Incomplete

Unfortunately, that is all that analysis that has been completed for the Byzantine tolerant algorithm. A complete analysis would finish the characterization of the transitive consistency check, then proceed to characterize the synchronization capabilities using the consistency checks.

Chapter 5

Experimental Application

A control system to balance an inverted pendulum was built to run on the SBFTC to demonstrate its fault tolerant capabilities. The application provides control commands to a motor that drives a horizontal, radial arm either clockwise or counterclockwise. The radial arm may move freely in a full circle. The pendulum is attached to the end of the radial arm and can swing freely in a full vertical circle, perpendicular to the radial arm. The job of the controller is to balance the pendulum upright while keeping the radial arm in front and centered. Appropriate control commands are determined by sensor data from two potentiometers measuring the angle of the pendulum and the radial arm. Figure 5-1 depicts the demonstration setup.

The experimental setup depicted in the figure is actually simulated by computer. A Simulink simulation of the dynamic system runs on the ICP, accepting control commands, updating the state of the system, and returning the new "sensor" data. Periodically, the ICP forwards the current state of the system to a graphical display running on a Windows machine. A user interface running on the Windows machine also allows a human to view the state of each FCP, manually inject faults into the fault-tolerant system, and "bump" the simulated pendulum. Implementing the demonstration as a simulation allowed the

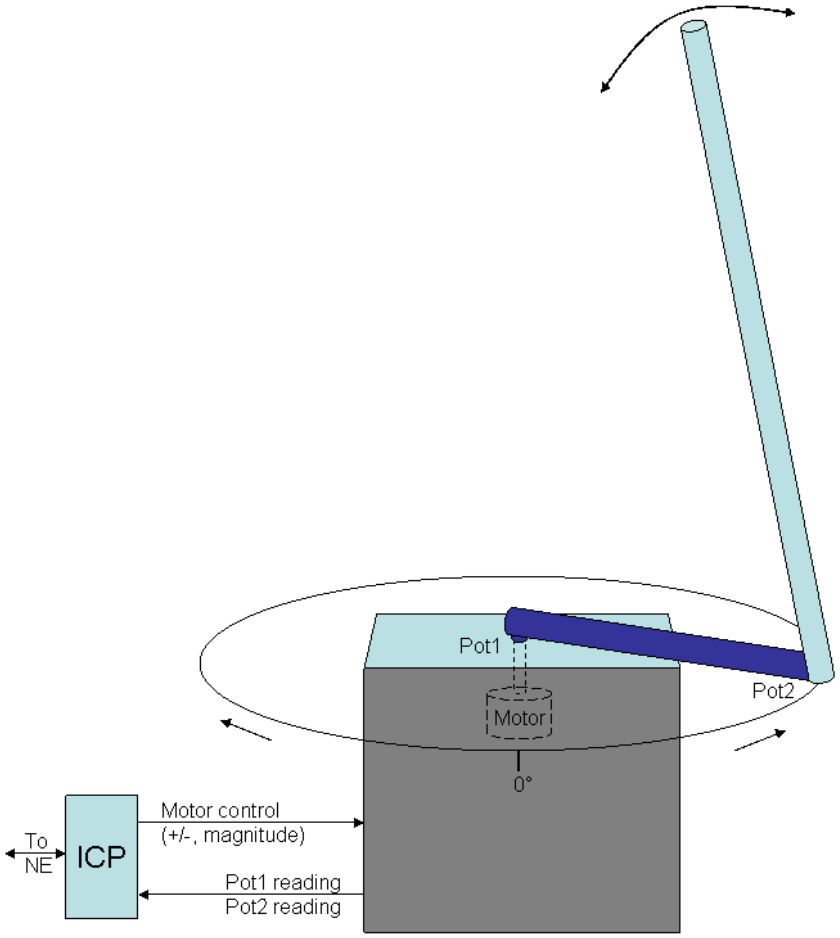

Figure 5-1: The inverted pendulum sits atop a radial arm extending from the center of a base. The pendulum should remain upright with the radial arm at the 0° mark unless disturbed by an external force, after which the system should recover. The radial arm is controlled by a motor which can push the arm in the clockwise and counterclockwise directions. Control is based on data from two potentiometers, which measure the angles of the pendulum and radial arm. The ICP acts as the interface to the motor and sensors, relaying information to and from the FCPs via its NE.

slowdown of time, which was necessary because the SBFTC proved unable to provide a control loop with a sufficiently high frequency. Figure 5-2 depicts the physical setup of the experiment and shows how software is distributed in the system. Figure 5-3 shows a screen shot of the experiment.

The demonstration in an example of a closed control loop. Control commands are sent from the FCPs to the simulated motor on the ICP, which drives the radial arm. In response, the ICP sends new potentiometer data back to the FCPs. The FCPs evaluate the data in preparation to issue the next command, closing the loop. The faster the data may travel back and forth between the FCPs and the ICP, the better the system will be able to balance the pendulum.

In the demonstration system, packets are sent between nodes every 8 ms, meaning that a class 1 message takes 16 ms to travel to its destination in a fault-tolerant manner and a class 2 message takes 8 ms. Since each cycle of the control loop requires a class 1 and a class 2 message, a full loop requires at least 24 ms, assuming that a control message or sensor message is ready to be sent at every communication cycle. Unfortunately, this is not the case. The NE's communication loop naively checks for messages from the FCP/ICPs immediately after it sends data to them. Therefore, the communication loop only learns about a response message in the following cycle, and the response can only be sent out in the cycle after that. Thus, a cycle is wasted at each end of the control loop. This translates to a control period of 40 ms and a frequency of 25 Hz. Figure 5-4 offers an approximate timing diagram of single control cycle.

Unfortunately, 25 Hz is not enough to successfully balance an inverted pendulum system with the specifications given earlier. In fact, to reliably balance the pendulum, it was necessary to slow simulation time down by a factor of 6, in effect raising the control loop frequency to 150 Hz.

The demonstration system used the failstop synchronization algorithm described in Sec-

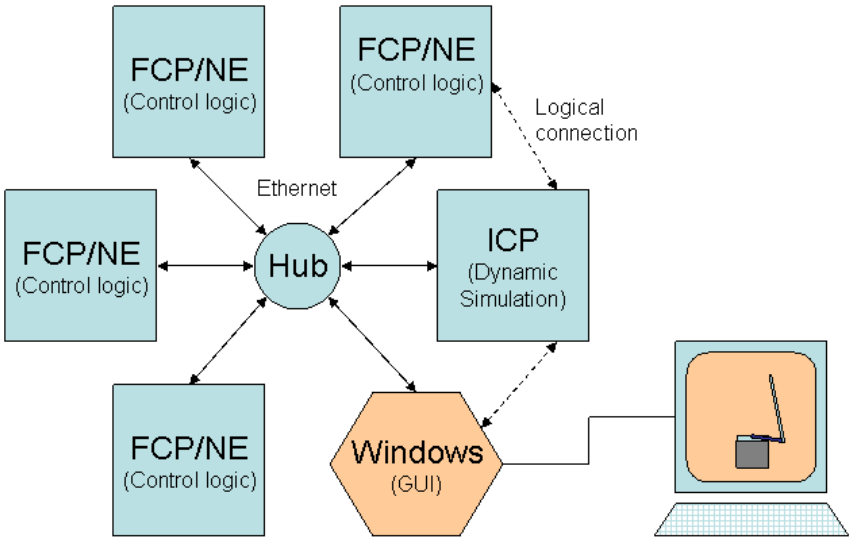

Figure 5-2: The physical components of the experimental application. Each FCP executes the control logic and sends the resulting motor commands to the ICP running the Simulink dynamic simulation software. The simulation software updates the state of the simulation and produces the next set of potentiometer readings, which are sent back to the FCPs. Periodically, the simulation state information is sent by the ICP to the Windows machine, which renders a graphical depiction of the pendulum. The Windows machine also runs a simple user interface, which allows a person to interact with the system.

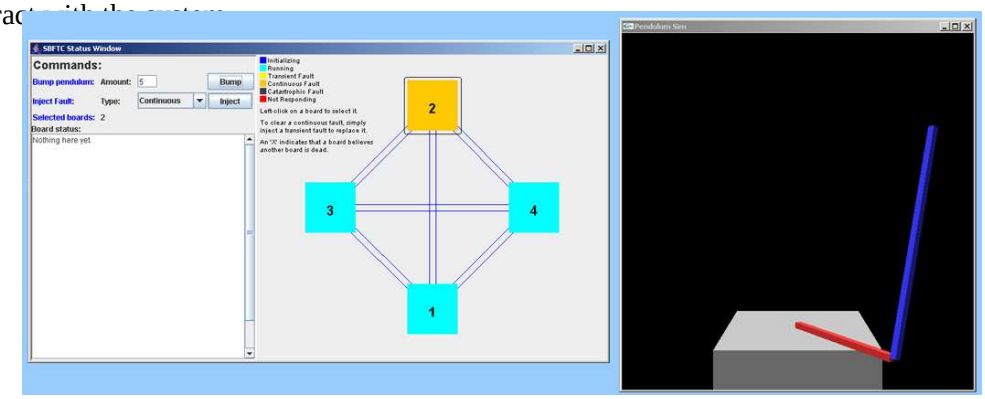

Figure 5-3: A screen shot of the SBFTC experimental application. The Pendulum Sim window depicts the current state of the pendulum simulation running on the ICP. The SBFTC Status Window panel allows a user to see the current state of the NE/FCPs, inject faults of varying severity into the system, and bump the pendulum in either direction. In this particular demonstration, node 2 is suffering from a “continuous” fault, meaning it is consistently corrupting the contents of messages it sends to other NEs. The graphical renderer of the pendulum was created by David Chau.

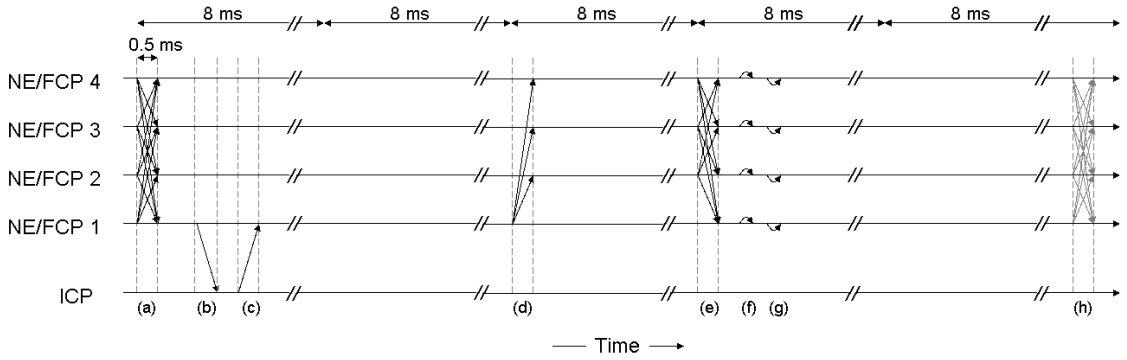

Figure 5-4: Visualization of demonstration application timing. Not to scale. Arrows between horizontal lines indicate messages being sent from one node to another. (a) NEs perform the 1st step of class 2 exchange with motor control data. (2nd step not pictured) (b) After voting and other processing, NE 1 sends control data to ICP. (c) After updating simulation state, ICP sends new sensor data to NE 1. (d) After recognizing the arrival of the sensor data in the previous round, NE 1 initiates a class 1 exchange. (e) All NEs wait until next communication round, then complete the class 1 message exchange. (f) After voting, each NE forwards sensor data to the FCP (on the same node). (g) After processing, FCPs sends new command to its respective NE. (h) After recognizing the arrival of the control data in the previous round, the NEs perform the 1st step of a class2 exchange, repeating the cycle. The communication medium has approximately a 500 μs delay. Communication rounds occur approximately once every 8 ms. This diagram is slightly misleading, since NEs do not all send their messages at the same time. In fact, the imperfect synchronization is a large part of why communication rounds are so far apart.

tion 4.3, since the Byzantine resilient algorithm has not yet been developed sufficiently to be correctly implemented.

Chapter 6

Results and Conclusions

The previous sections have described the design of the SBFTC, provided implementation details, and described the experimental application built to demonstrate the system. This section describes some tangible results that were achieved during that work and provides an evaluation of the results and recommendations based on that work. The section ends with a summary of the thesis and recommendations for future work in continuance of this thesis.

6.1 Prototype and Experimental Results

The section begins by evaluating the current capabilities of the prototype SBFTC. Next, the performance of the synchronization algorithms are discussed. Since the failstop algorithm has been fully characterized, some theoretical and actual performance data is presented.

6.1.1 Current SBFTC Limitations

The prototype SBFTC handles only a limited number of fault scenarios. Most importantly, it correctly handles any single FCP/NE which produces inconsistent output and/or corrupts

the message bodies from other NEs by detecting and eliminating the inconsistency via voting. Corrupted message headers are recognized, but they are not handled in a consistent way. It is assumed that only a single node is faulty at any one time.

The system does not currently handle omitted messages correctly. A node that does not receive a message from another node at the expected time just assumes that the offending node is faulty. The honest nodes do not communicate to make a consistent decision if a node is faulty. If a node fails to send a message to only a subset of the other nodes, only a subset of the correct nodes might believe the bad node is faulty, and inconsistent behavior may result.

The SBFTC does not tolerate faulty nodes during startup. All four NEs are required to be available for initialization. The SBFTC also does not handle reintegration of a manually-reset node into a running system. Once a node has suffered a major fault, it cannot rejoin.

The SBFTC was designed with four NEs instead of five, like the X-38, since the goal is to eventually implement a cryptographic signature scheme for messages. The SBFTC does not currently implement such signatures. For research performed toward this goal, see [1].

There is currently a significant, intermittent bug that causes nodes to stop responding all together until the power is cycled. The most likely cause of this bug is the implementation of the synchronization algorithm, which timestamps UDP packets when they are received by a node. Currently, the packets are timestamped by reading the system clock within the interrupt routine that handles incoming packets, which may cause kernel-level deadlock.

6.1.2 Synchronization Algorithm Performance

The failstop synchronization algorithm presented in this thesis manages to achieve synchronization smaller in magnitude than the maximum network latency. It manages to do so by incorporating knowledge of the minimum network latency as well. The achievable

synchronization is largely a function of the spread between the minimum and maximum network latencies.

Using equations (4.18) and (4.19) and reasonable values for the required constants, the graphs in figures 6-1 and 6-2 emphasize that latency spread is the primary factor in achievable synchronization.

The Byzantine resilient version of the algorithm, though still not fully developed, appears to be an interesting candidate. Like the failstop algorithm, it will still be highly dependent on the predictability of the network latency, but in the controlled environment of an embedded system running an RTOS, it appears that the network latency can be tightly controlled, allowing the algorithm introduced in this paper to be used effectively.

Take note, however, that the achievable D_{max} for a Byzantine resilient implementation will be much less favorable than that achievable for failstop implementation. In addition to requiring a much longer period of time to execute, reducing possible synchronization, an adversary may cause nodes to set their clocks unnaturally far forward, causing temporary clock skews between nodes to be more than twice as large as possible without an adversary. However, if network latency is sufficiently predictable, this limitation could be managed.

More development of the algorithm needs to be performed, not just to fully characterize the Byzantine resilient algorithm, but also to describe how an initialization strategy might work. The analysis done in this thesis assumes that all clocks begin with some bounded skew, but it is difficult to make such an assurance on startup. Currently, the prototype implementation uses Optimal Clock Sync to reach an acceptable level of synchronization before the algorithm described in this paper takes over to maintain the same synchronization.

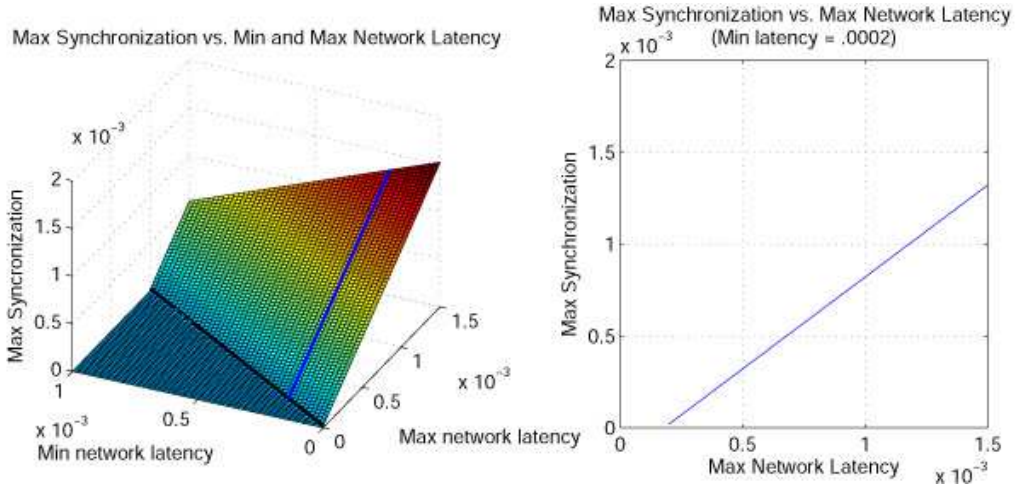

Figure 6-1: Achievable synchronization with $\rho = 10^{-5}$, $dt_{calc_{max}} = 50\mu s$, and $dt_{between} = 1s$. The second graph corresponds to the thick line running up the slope in the first graph. Note that when the minimum latency is close to the maximum latency, the achievable synchronization becomes very small. The flat and level region is where $\ell_{min} > \ell_{max}$ and D_{max} is undefined.

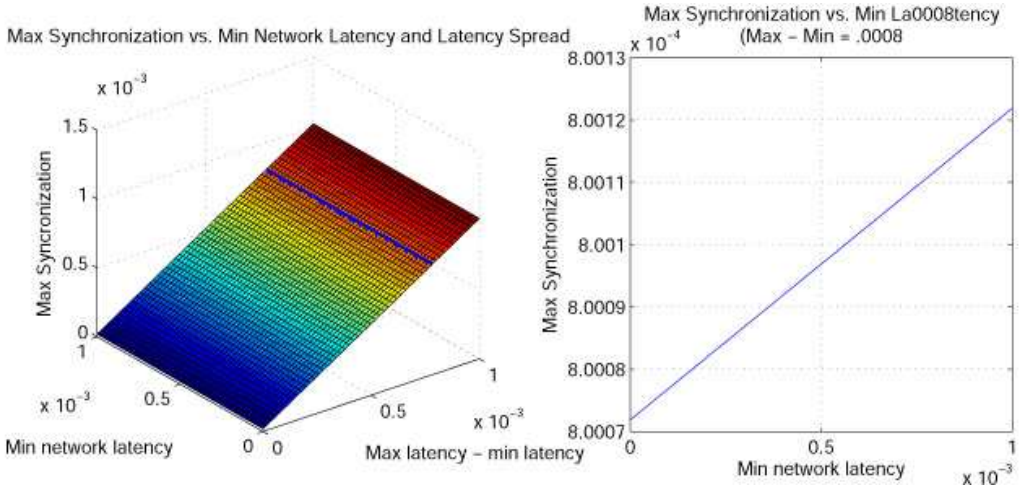

Figure 6-2: These graphs are taken from the same data as figure 6-1. The first emphasizes that the achievable synchronization is largely a function of the latency spread rather than the actual latency. The second graph corresponds to the thick line running perpendicular to the slope in the first graph. It demonstrates the second order effect of absolute network latency in determining achievable synchronization.

6.1.3 SBFTC Performance

The prototype SBFTC is not as successful as hoped, since it is unable to balance a simulated inverted pendulum in real time. The most critical reason for the performance failure is communication latency. As described in Chapter 5, messages are sent between NEs only once every 8 ms, much too slowly. The long delay between messages ensures that every NE has completed the work required in the previous communication loop before it must send another message.

There are several reasons why a single communication loop takes so long, one of which is the point-to-point latency. Of course, large latencies directly affect the communication loop frequency, since messages may not be sent as quickly. However, large latencies also indirectly affect the system in a more dramatic way by affecting the level of achievable synchronization. As noted in Section 6.1.2, uncertain latency, not large latency, is the primary factor in unfavorable clock synchronization. However, since the nodes share a common network, there is much variation in the latency, and the variation is only aggravated by large maximum latencies. Hence, the system suffers from poor synchronization. The farther spread apart clocks are, the longer a node's communication loop must wait to ensure all other nodes have completed their own loop before continuing on to the next cycle.

One reason for the large latency is that the NE communication layers were built on top of the UDP protocol. UDP is a transport-layer protocol, designed to move packets over a large network with routes consisting of several hops. Since NE communication is point-to-point, UDP gives no advantage over the lower-level, more efficient ethernet protocol that runs at the link layer.

In the SBFTC prototype, UDP packets are approximately 1 KB in size. A one-way trip of a 1 KB UDP packet over the prototype SBFTC's empty network takes about 400 μs. Also recall that several communication layers were built on top of the UDP protocol

that also contribute to the time spent in the network stack. A message sent at the virtual processor (VP) layer (NE-to-NE) takes approximately 500 μs to travel one-way, about 100 μs longer than if the message were sent at the transport layer. Furthermore, a message sent from an NE to an FCP on the same physical node takes approximately 250 μs. These three values imply that during a trip across the network, a packet spends 100 μs in the SBFTC network layers, 150 μs in the lower communication layers, and 250 μs on the actual wire. These values are summarized in table 6.1.

Communication Layer(s)	**Typical Latency (μs)**
Physical layer (wire)	250
Link, network, transport layers	150
SBFTC board, processor layers	100

Table 6.1: Latencies of Network Layers for a 1 KB Packet

More significant than the actual maximum network latency is the uncertainty in the possible spread of network latencies. As demonstrated in Section 6.1.2, the possible synchronization is more affected by the unpredictability of the network than by the absolute latency of the network. Unfortunately, all the nodes of the prototype SBFTC share a common network, and therefore the network latency is very unpredictable due to wire contention.

The message processing components of the communication loop take an encouragingly short amount of time. Voting the different versions of a single class 1 or class 2 message, which consists of three memory buffer comparisons of 128 bytes, takes approximately 30 μs, and the processing required to execute a single round of clock synchronization takes approximately 50 μs. The major time sink of the loop tends to be the overhead required by the network stacks when sending and receiving messages to NEs, ICPs, and FCPs, which may occur several times per loop. Table 6.2 lists the different functions performed by the communication loop and the various associated execution times.

Clearly, the CPU time spent in the communication stack reading and writing messages

Function	**Approximate Execution Time (μs)**
Setup before sending. Examine messages table for messages ready to be sent.	20
Send appropriate messages to other NEs. (3 1312 byte buffers)	1370
Wait to ensure messages from other NEs arrive. (Pre-specified sleeping period)	3200
Read arrived NE messages from memory. (3 1312 byte buffers)	250
Parsie messages and update active messages table.	500
Compare arrival time of messages and adjusting clock. (periodic)	35
Vote a single group of messages versions. (3 128 byte buffer comparisons)	30
Forward appropriate messages to ICP/FCP	500
Receive messages from ICP/FCP	60

Table 6.2: Communication Loop Functions and Execution Times

takes up the majority of the time during the execution of a single cycle, although what parts of the stack are time consuming is not immediately clear. If a method was found to slash these times, the frequency of the communication loop could be increased substantially. Why parsing messages and updating the active messages table takes so long has not been analyzed, but it is likely that inefficient algorithms, like full table searches, are responsible.

Another problem with the communication loop is that the FCP/ICPs are polled for messages at a rather poor time. When a message is sent from an FCP/ICP to an NE, the message is received by the NE's hardware and stored in a buffer. The NE chooses when to poll that buffer for any waiting messages. In the current communication loop, the buffer is polled only once per loop, directly after any voted messages are sent to the processor.

The FCP/ICP may receive a message from the NE, complete its processing, and send a reply back to the NE in time for the next communication cycle. However, the NE will not poll its buffer and discover the waiting message until the *end* of the next loop, after

that round's messages have already been sent. As a result, that cycle is wasted and the message is forwarded in the following cycle. A more intelligently timed poll (just before NE messages are sent) could eliminate two wasted cycles and 16 ms from the control period, increasing the control frequency by about 38%.

6.1.4 Design Alternatives

A critical design decision in this prototype SBFTC was to pass messages between NEs only at pre-specified times. This decision was made because of the ease of implementation and predictability. The major drawback is that message passing between virtual groups is inefficient, since each step of the fault-tolerant algorithm requires a complete cycle of the communication loop.

An alternative design would have been event driven, with NEs quickly reacting to messages arriving from their FCP/ICPs and other NEs and immediately forwarding or reflecting them. Such a design would certainly reduce message latency across the fault-tolerant network, but would likely be more difficult to correctly design and analyze. For example, given that messages from FCPs should have the lowest latency possible, if an NE receives a message to forward from its ICP, it needs to make a decision whether to forward it immediately, wait for an expected FCP packet, defer to a synchronization routine that might begin soon, etc. Careful scheduling would be required to make the system predictable.

One downside to event driven programming is that it is non-deterministic. There is no way to predict exactly when a piece of code will be executed if it is triggered by an external event. If a CPU intensive task were executed as the result of some unpredictable event, that code might usurp another important task in a dangerous way. Thus, event driven programming is typically ill-suited for critical applications.

An alternative to event driven programming is polling. While the current SBFTC pro-

totype already uses polling to determine whether messages have arrived from other NEs or FCP/ICPs, in order to approach the response times of event driven programming, the frequency of polling would need to be much higher. Unfortunately, polling is a CPU intensive process. Since NEs and FCPs reside on the same node, a constantly polling NE might starve the FCP of CPU time. Scheduling the FCP for the CPU between polling could impose large overhead from context switches. Reducing the frequency of polling increases available CPU, but decreases response times.

6.2 Discussion and Recommendations

This section discusses conclusions and recommendations arising from this thesis.

General Design

In general, the periodic, synchronized communication loop was an intuitive and easily implemented design. It was chosen after a fellow student ran into difficulties designing an event driven system. If the difficulties with large communication latencies are solved, this design will become quite feasible.

The choice of putting the FCP on board with the NE in an interesting one, since it cut communication latency between the NE and the FCP by about half. However, doing so requires careful analysis of the execution time required by the FCP, since it no longer has the full resources of its hardware available to it. Due to the time-partitioning offered by Integrity, the FCP will never be unexpectedly starved, but the system designer must treat the NE partition with priority (or else the system will fail), and the FCP may not get the proper resources to complete in a timely manner. If communication latencies are slashed and the communication loop runs at a much higher frequency, its possible that the NE

process will require all of the CPU to keep up. In this case, the FCP will be better moved to a different node. Communication between the NE and FCP would be increased, but it would be better than giving a 30 μs FCP process only 10 μs every 100 μs to execute.

Software Design

During the development of the SBFTC prototype, not much care was taken to optimize code paths and data structures. This choice turned out to be acceptable, since the majority of execution time is spent during communication. However, once the high communication time is solved, more attention should be paid to optimizing the actual message processing routines.

Communication Network

This thesis has demonstrated that communication latency and predictability, both on the wire and in the software stack, are the major limiting factors in the current SBFTC system. Effort should be put into redesigning the existing physical layout to support point-to-point NE communications and backplane communications with ICPs. Removing the network and transport layers from the current communication stack, and optimizing the SBFTC communication layers should be a priority.

Synchronization

The original synchronization algorithm suggested for this system was the Optimal Clock Synchronization. However, given that minimal drift rate from real-time is not specified as a requirement, algorithms like the one presented here, for which agreement properties depend on latency variation rather than absolute latency, are attractive candidates, given the

potentially highly-predictable network.

The algorithm presented here may be substituted with the more efficient (and fully analyzed) algorithm presented in [5], but care must be taken to correctly handle clocks that are set backwards. One solution is to amortize the clock adjustment over a large period of time, as discussed in [4], but this requires building a level of indirection to access the corrected system clock.

The advantage of the algorithm presented here is that no such amortization is needed to ensure that clocks are not set backwards. However, it should be noted that the large adjustments forward may still require amortization to smooth the clock progression, depending on how the SBFTC is implemented, and thus, the presented algorithm may not provide any advantage over [5].

6.3 Summary and Future Work

This thesis described the design of a prototype software-based fault tolerant computer that can mask errors that may arise during runtime. The ultimate goal of the project is to replace the X-38 fault tolerant computer, which requires proprietary hardware, in favor of a software-based solution that may run on cheap, commercially-available hardware. The current prototype implements a basic system of four nodes which can support a four-redundant logical computer, synchronize each redundant processor, and handle input and output to and from the redundant nodes in a Byzantine fault-tolerant way.

This thesis also analyzed a failstop synchronization algorithm that infers clock skews based on the arrival times of expected messages from other nodes and guarantees that clocks are never set backward. An extension to the algorithm to provide Byzantine fault-tolerance is proposed and described, but the analysis is not complete. Both algorithms are well-suited for embedded systems, since the achievable synchronization is heavily depen-

dent on the predictability of the communication network between the nodes. The non-fault-tolerant algorithm was used by the prototype SBFTC.

A demonstration application was built that balanced a simulated inverted pendulum and could handle corrupted messages produced by a single node. The demonstration was not a complete success, since the prototype SBFTC proved incapable of supporting the control frequency required to balance the pendulum unless the simulation was slowed by a factor of six.

There are several results presented in this thesis that could be improved. First, the prototype SBFTC needs to be completed. This includes extending the startup routine for real-time virtual group configuration, extending fault handling to enforce consistent behavior among honest nodes in the event of a fault, setting up true point-to-point communication between the nodes, and relocating the ICP so that it no longer communicates with its NE over the common network.

Beyond such major changes, the SBFTC may be improved in other ways. The communication loop of the prototype SBFTC could be dramatically optimized. For example data structures and associated code can be improved, particularly with regards to the active message table. Also, the timing of message sending could be better analyzed to reduce the amount of time each node waits for messages from other nodes. Another improvement could be more intelligent scheduling of polling for messages from ICPs and FCPs.

The synchronization algorithms could be better developed. Clearly, the Byzantine-resilient algorithm needs to be fully analyzed, but there are other needs as well. For example, no initialization strategy to synchronize nodes at startup is presented in this thesis. Also, no analysis is done on the rate of drift of the synchronized clocks from real time. Clocks drift naturally from real time, however, the forward adjustment of the clocks as part of the synchronization routine could cause the synchronized clocks to drift faster than the natural rate. This effect should be better understood.

The synchronization algorithms might also be optimized. Currently, all nodes wait for the arrival of other messages after they have sent their own. However, it might not be necessary for the trailing node to wait at all. A trailing node would know it was last because it already received messages from all other nodes before it sent its own. Thus the amount of time required for the algorithm to complete would be reduced, increasing the achievable synchronization.

Other future work might include removing the network and transport layers from the SBFTC communication stack.

6.4 Acknowledgements

This research and development is being conducted at Charles Stark Draper Laboratory as an internal research and development project under the supervision of Roger Racine, with additional oversight by Professor Barbara Liskov of the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). The author would like thank both for their help and guidance. In addition, the author would also like to thank Samuel Bailin of Draper Laboratory for imparting his knowledge of the X-38 fault-tolerant computer. Finally, thanks is most earned by David Chau, fellow Draper Fellow, MIT student and collaborator, for his superior insight and hard work toward the successful completion of this thesis. The author also wishes to thank his mommy.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] David Chau. Authenticated messages for a fault-tolerant computer system. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2006.

[2] Danny Dolev, Joseph Y. Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the Association for Computing Machinery*, 42(1):143–185, 1995.

[3] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. In *ACM Transactions on Programming Languages and Systems*, volume 4, pages 382–401, July 1982.

[4] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the Association for Computing Machinery*, 32(1):52–78, 1985.

[5] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 75–88, New York, NY, USA, 1984. ACM Press.

[6] Kevin Marzullo. *Loosely-Coupled Distributed Services: A Distributed Time Service*. PhD thesis, Stanford University, Stanford, CA, 1983.

[7] David L. Mills. *Computer Network Time Synchronization: the Network Time Protocol*. CRC Press, 2006.

[8] Roger Racine, Michael LeBlanc, and Samuel Beilin. Design of a fault-tolerant parallel processor. In *Proceedings of the Digital Avionics System Conference*, volume 2, pages 13D2–1–13D2–10, 2002.

[9] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the Association for Computing Machinery*, 34(3):627–645, July 1987.