MIT/LCS/TR-335

SYNCHRONIZING CLOCKS IN A DISTRIBUTED SYSTEM

Jennifer Lundelius

Synchronizing Clocks in a Distributed System

by

Jennifer Lundelius

B.A., The University of Texas at Austin
(1979)

Submitted to the
Department of Electrical Engineering
and Computer Science
in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August, 1984

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
August 17, 1984

Certified by ______________________________
Nancy A. Lynch, Thesis Supervisor

Accepted by ______________________________
Arthur C. Smith, Chairman, Departmental Committee on Graduate Students

Synchronizing Clocks in a Distributed System

by

Jennifer Lundelius

Submitted to the
Department of Electrical Engineering and Computer Science
on August 17, 1984 in partial fulfillment of the requirements
for the Degree of
Master of Science in Computer Science

Abstract

Keeping the local times of processes in a distributed system synchronized in the presence of arbitrary faults is important in many applications and is an interesting theoretical problem in its own right. In order to be practical, any algorithm to synchronize clocks must be able to deal with process failures and repairs, clock drift, and varying message delivery times, but these conditions complicate the design and analysis of algorithms. In this thesis, a general formal model to describe a system of distributed processes, each of which has its own clock, is presented. The processes communicate by sending messages to each other, and they can set timers to cause themselves to take steps at some future times. It is proved that even if the clocks run at a perfect rate and there are no failures, an uncertainty of ε in the known message delivery time makes it impossible to synchronize the clocks of n processes any more closely than $2\varepsilon(1 - 1/n)$. A simple algorithm that achieves this bound is given to show that the lower bound is tight.

Two fault-tolerant algorithms are presented and analyzed, one to maintain synchronization among processes whose clocks initially are close together, and another to establish synchronization in the first place. Both handle drift in the clock rates, uncertainty in the message delivery time, and arbitrary failure of just under one third of the processes. The maintenance algorithm can be modified to allow a failed process that has been repaired to be reintegrated into the system. A variant of the maintenance algorithm is used to establish the initial synchronization. It was also necessary to design an interface between the two algorithms since we envision the processes running the start-up algorithm until the desired degree of synchronization is obtained, and then switching to the maintenance algorithm.

Keywords: synchronization, clocks, distributed systems, fault tolerance.

Thesis supervisor: Nancy A. Lynch
Title: Associate Professor of Computer Science and Engineering

Acknowledgments

I would like to thank my advisor, Nancy Lynch, for contributing so many of her ideas and so much of her time to this thesis. Without her technical guidance and constant interest it would have taken me infinitely longer to finish.

Brian Coan, Cynthia Dwork, Gene Stark, and Bill Weihl kindly read parts of this document and provided valuable suggestions and criticisms.

I am grateful to many people for their friendship and moral support during the last two years, especially Randy Forgaard for expeditions to Toscanini's, Ron Kownacki for his healthy psychological equilibrium when we were TA's, Brian Oki for sound advice about the lab, Jim Restivo for reciprocating complaints, Lori Sorayama for being a sympathetic listener, Kathy Yelick for letting me bounce incoherent ideas off her, Joe Zachary for his unique brand of humor, and the regular Friday-afternoon-at-the-Muddy crowd.

Most importantly, my mother and father were an unceasing source of encouragement, patience, and love.

Table of Contents

Table of Figures

Chapter One

Introduction

1.1 The Problem

Keeping the local times of processes in a distributed system synchronized in the presence of arbitrary faults is important in many applications and is an interesting problem in its own right. In order to be practical, any algorithm to synchronize clocks must be able to deal with process failures and repairs, clock drift, and varying message delivery times, but these conditions complicate the design and analysis of algorithms.

In this thesis we describe a formal model for a system of distributed processes with clocks, and demonstrate a lower bound on how closely the clocks can be synchronized, even when strong assumptions are made about the behavior of the system. Then we describe and analyze algorithms to establish and maintain synchronization under more realistic assumptions.

We assume a collection of processes that communicate by sending messages over a reliable medium. Each process has a physical clock, not under its control, that is incremented in some relationship with real time. By adding the value of a local variable to the value of the physical clock, the process obtains its local time.

The design of a clock synchronization algorithm must take into account the following factors.

1. The uncertainty in the message delivery time. Messages are assumed in this thesis to be delivered a fixed amount of time after they are sent, plus or minus some uncertainty.

2. Clock drift. Are the processes' clock rates fast or slow relative to real time? If the clocks drift, then the synchronization procedure must be repeated periodically to keep the clocks synchronized.

3. Are the clocks initially synchronized? If they are, then the problem of synchronizing the clocks is already solved unless the clocks drift, since once nondrifting clocks are synchronized, they stay synchronized.

4. Fault tolerance. What kinds of faults (if any) are tolerated? This thesis does not consider communication link failures. A certain proportion of the processes, however, may be faulty in the worst possible way, by sending arbitrary messages at arbitrary times.

5. Digital signatures. Can a faulty process forge a message from another process? If digital signatures are available, then process p can tell process q that it received a message x from process r, only if such was actually the case. This obviously reduces the power of a faulty process to create havoc. Some of the other clock synchronization algorithms in the literature [5, 7] need this capability, but ours do not.

6. Reintegration. In order to be practical, a synchronization algorithm must allow faulty processes that have recovered to be reintegrated into the system.

7. Size of the adjustment. Particularly when the synchronization procedure is performed periodically, the amount by which the clock is changed should not be too big.

1.2 Results of the Thesis

1.2.1 Model

One of the contributions of this thesis is a precise formal model of a system of distributed processes, each of which has its own clock. Within the model, lower bound proofs can be seen to be rigorous, and the effects of algorithms, once they are stated in a language that maps to the model, can be discerned unambiguously. The model is described in Chapter 2.

We model the situation in which each process has a physical clock that is not under its control. By adding some value to the physical clock time a process obtains a local time. A process can set a timer to go off at a specified time in the future. Formally, timers are treated similarly to messages between processes. The system is interrupt-driven in that a process only takes a step when a message arrives. The message may come from another process, or it may be a timer that was set by the process itself. Thus, by using a timer, a process can ensure that an interrupt will occur at a specified time in the future.

A process is modelled as an automaton, with states and a transition function. One of the arguments to the transition function is a real number, representing the time on the process' clock. Clocks are modelled as real-valued functions from real time to clock time. We assume that the communication network is fully connected, so that every process can send a message directly to every other process. Processes possess the capability of broadcasting a message to all the processes at the same time. The message system is described as a buffer that holds messages until they are delivered. All messages are delivered within a fixed amount of time plus or minus some uncertainty. The delivery of a message at a process is the only type of event we consider. A system execution consists of sequences of "actions", each of which is a process event surrounded by a description of the state of the system, one sequence for each real time of interest. The sequences must satisfy certain natural consistency and correctness conditions.

1.2.2 Lower Bound

Even if the simplifying assumptions are made that clocks run at a perfect rate and that there are no failures, the presence of an uncertainty of ε in the message delivery time alone prevents any algorithm from exactly synchronizing clocks that initially have arbitrary values. We show in Chapter 3 that $2\varepsilon(1 - 1/n)$ is a lower bound on how closely the clocks of n processes can be synchronized in this case. Of course, in this case, any algorithm which synchronizes the clocks once causes them to remain synchronized. However, since these are strong assumptions, this lower bound also holds for the more realistic case in which clocks do drift and arbitrary faults occur. Just to show that this bound is tight, we describe an algorithm that achieves this bound for the simplified case.

1.2.3 Maintaining Synchronization

We describe a synchronization algorithm in Chapter 4 that handles clock drift, uncertainty in the message delivery time and arbitrary process faults. The algorithm requires the clocks to be initially close together and less than one third of the processes to be faulty.

Our algorithm runs in rounds, resynchronizing every so often to correct for the clocks drifting out of synchrony, and using a fault-tolerant averaging function based on those in [1] to calculate an adjustment. The size of the adjustment made to a clock at each round is independent of the number of faulty processes. At each round, n^2 messages are required, where n is the total number of processes. The closeness of synchronization achieved depends only on the initial closeness of synchronization, the message delivery time and its uncertainty, and the drift rate. Since the closeness of synchronization depends on the initial closeness, this is, in the terminology of [7], an *interactive convergence* algorithm. We give explicit bounds on how the difference between the clock values and real time grows. The algorithm can be easily adapted to become a reintegration procedure for repaired processes.

At the beginning of each round, every nonfaulty process broadcasts its clock value and then waits a bounded amount of time, measured on its logical clock, long enough to ensure that clock values are received from all nonfaulty processes. After waiting, the process averages the arrival times of all the messages received, using a particular fault-tolerant averaging function. The resulting average is used to calculate an adjustment to the process' clock.

The fault-tolerant averaging function is derived from those used in [1] for reaching approximate agreement. The function is designed to be immune to some fixed maximum number, f, of faults. It first throws out the f highest and f lowest values, and then applies some ordinary averaging function to the remaining values. We choose the midpoint of the range of the remaining values, to be specific. The properties of the fault-tolerant averaging function allow the distance between the clocks to be halved, in a rough sense, at each round. Consequently, the averaging function can be considered the heart of the algorithm.

This algorithm can maintain a closeness of synchronization of approximately 4ε, where ε is the uncertainty in the message delivery time.

1.2.4 Establishing Synchronization

The problem solved by the algorithm in Chapter 4 is only that of maintaining synchronization of local times once it has been established. There is, of course, the separate problem of establishing such synchronization in the first place among processes whose clocks have arbitrary values. A variant of the maintenance algorithm can be used to establish the initial synchronization as well and is described in Chapter 5. The algorithm handles arbitrary failures of the processes, uncertainty in the message delivery time, and clock drift. It was also necessary to design an interface between the two algorithms since we envision the processes running this algorithm until the desired degree of synchronization is obtained, and then switching to the maintenance algorithm.

The structure of the algorithm is similar to that of the algorithm which maintains synchronization. It runs in rounds. During each round, the processes exchange clock values and use the same fault-tolerant averaging function as before to calculate the corrections to their clocks. However, each round contains an additional phase, in which the processes exchange messages to decide that they are ready to begin the next round.

This algorithm also synchronizes the clocks to within about 4ε. Again, the fault-tolerant averaging function used in the algorithm causes the difference in the clocks to be cut in half at each round.

1.3 Related Work

The problem of synchronizing clocks has been a topic of interest recently. A seminal paper was Lamport's work [6], defining logical clocks and describing an algorithm to synchronize them. Several algorithms to synchronize real time clocks have appeared in the literature [5, 6, 7, 9]. Those of Lamport [6] and Marzullo [9] have the processes updating their clocks whenever they receive an appropriate message; these messages are assumed to arrive every so many real seconds, or more often. In contrast, the algorithms in Halpern, Simons and Strong [5], Lamport and Melliar-Smith [7], and this thesis run in rounds. During a round, a process updates its clock once. The rounds are determined by the times at which different processes' local clocks reach the same times. There is an impossibility result due to Dolev, Halpern and Strong [2], showing that it is impossible to synchronize clocks without digital signatures if one third or more of the processes are subject to Byzantine failures. Dolev, Halpern and Strong's paper [2] also contains a lower bound similar to ours (proved independently), but characterizing the closeness of synchronization obtainable along the real time axis, that is, a lower bound on how closely in real time two processes' clocks can read the same value.

The three algorithms of Lamport and Melliar-Smith [7], as well as our maintenance algorithm, require a reliable, completely connected communication network, and handle arbitrary process faults. The first algorithm works by having each process at every round read all the other processes' clocks and set its clock to the average of those values that aren't too different from its own. The size of the adjustment is no more than the amount by which the clocks differ plus the uncertainty in obtaining the other processes' clock values. However, the closeness of the synchronization achieved depends on the total number of processes, n. The message complexity is n^2 at each round, if getting another process' clock value is equated with sending a message.

In the other two algorithms in [7], each process sets its clock to the median of the values obtained by receiving messages from the other processes. To make sure each nonfaulty process has the same set of values, the processes execute a Byzantine Agreement protocol on the values. The two algorithms use different Byzantine Agreement protocols. One of the protocols doesn't require digital signatures, whereas the other one does. As a result, the clock synchronization algorithm derived from the latter will work even if almost one half of the processes are faulty, while the other two algorithms in [7] can only handle less than one third faulty processes. For both of the Byzantine clock synchronization algorithms, the closeness of synchronization and the size of the adjustment depend on the number of faulty processes, and the number of messages per round is exponential in the number of faults.

The algorithm of Halpern, Simons and Strong [5] works in the presence of any number of process and link failures as long as the nonfaulty processes can still communicate. It requires digital signatures. When a process' clock reaches a certain value (decided on in advance), it broadcasts that time. If it receives a message containing the value not too long before it reaches the value, it updates its clock to the value and relays the message. The closeness of synchronization depends only on the drift rate, the round length, the message delivery time, and the diameter of the communication graph after the faulty elements are removed. The message complexity per round is n^2. However, the size of the adjustment depends on the number of faulty processes.

The framework and error model used by Marzullo in [9] make a direct comparison of his results with ours difficult. He considers intervals of time and analyzes the error probabilistically.

The problem addressed in these papers is only that of maintaining synchronization of local times once it has been established. None of them explicitly discusses any sort of validity condition, quantifying how clock time increases in relation to real time. Only [5] includes a reintegration procedure for repaired processes.

Chapter Two

Formal Model

2.1 Introduction

We present a formal model for describing a system of distributed processes, each of which has its own clock. The processes communicate by sending messages to each other, and they can set timers to cause themselves to take steps at some specified future times. The model is designed to handle arbitrary clock rates, Byzantine process failures, and a variety of assumptions about the behavior of the message system.

The advantages of a formal model are that lower bound proofs can be seen to be rigorous, and the effects of an algorithm, once it is stated in a language that maps to the model, can be discerned unambiguously.

This model will be used in subsequent chapters to describe our particular versions of the clock synchronization problem.

2.2 Informal Description

We model a distributed system consisting of a set of processes that communicate by sending messages to each other. Each process has a physical clock that is not under its control.

A typical message consists of text and the sending process' name. There are also two special messages, START, which comes from an external source and indicates that the recipient should begin the algorithm, and TIMER, which a process receives when its physical clock has reached a designated time.

A process is modelled as an automaton with a set of states and a transition function. The transition function describes the new state the process enters, the messages it sends out, and the timers it sets for itself, all as a function of the process' current state, received message and physical clock time. An application of the transition function constitutes a process step, the only kind of event in our model.

The system is interrupt-driven in that a process only takes a step when a message arrives. The message may come from another process, or it may be a TIMER message that was sent by the process itself. Thus, by using a TIMER message, a process can ensure that an interrupt will occur at a specified time in the future. We neglect local processing time by assuming that the processing of an arriving message is instantaneous.

We assume that the communication network is fully connected, so that every process can send a message directly to every other process. Processes possess the capability of broadcasting a message to all the processes at one step. The message system is described as a buffer that holds messages until they are delivered.

System histories consist of sequences of "actions", each of which is a process event surrounded by a description of the state of the system, one sequence for each real time of interest. The sequences must satisfy certain natural consistency and correctness conditions. We introduce the notion of "shifting" the real times at which a particular process' steps occur in a history and note the resulting changes to the message delivery times. Finally, we define an execution to be a history in which the message system behaves as desired.

2.3 Systems of Processes

Let P be a fixed set of *process names*. Let X be a fixed set of *message values*. Then M, the set of *messages*, is {START, TIMER} $\cup$ (X x P). A process receives a START message as an external indication of the beginning of an algorithm. A process receives a TIMER message when a specified time has been reached on its physical clock. All other messages consist of a message value and a process name, indicating the sender of the message.

Let $\mathcal{F}(S)$ denote the finite subsets of the set S.

A *process* p is modelled as an automaton. It has a set Q of states, with a distinguished subset I of initial states, and a distinguished subset F of final states. It has a *transition function*, τ, where τ: Q x $\mathbb{R}$ x M $\rightarrow$ Q x $\mathcal{F}$(X x P) x $\mathcal{F}(\mathbb{R})$. The transition function maps p's state, a real number indicating its physical clock time, and an incoming message, all to a new state for p, a finite set of (message value, destination) pairs, and a finite set of times at which to set timers. For any r in $\mathbb{R}$, m in M, Y in $\mathcal{F}$(X x P), and Z in $\mathcal{F}(\mathbb{R})$, if q is in F and if τ(q,r,m) = (q',Y,Z), we require that q' also be in F. That is, once a process is in a final state, it can never change to non-final state.

We assume that, in the absence of non-TIMER messages, a process does not set an infinite sequence of timers for itself within a finite amount of time. To state this condition formally, we choose any time r_1 and state q_1 for p, and consider the following sequence of applications of τ_p:

$\tau_p(q_1,r_1,\mathrm{TIMER}) = (q_2,Y_2,Z_2)$

$\tau_p(q_2,r_2,\mathrm{TIMER}) = (q_3,Y_3,Z_3)$, where $r_2 = \min\{r \in Z_2: r > r_1\}$

.

.

.

$\tau_p(q_i,r_i,\mathrm{TIMER}) = (q_{i+1},Y_{i+1},Z_{i+1})$, where $r_i = \min\{r \in \cup_{j=2..i}Z_j: r > r_{i-1}\}$

.

.

.

Then as i approaches ∞, it must be that r_i approaches ∞.

We define a *step* of p to be a tuple (q,r,m,q',Y,Z) such that $\tau(q,r,m) = (q',Y,Z)$.

A *clock* is a monotonically increasing, everywhere differentiable function from $\mathbb{R}$ (real time) to $\mathbb{R}$ (clock time). We will employ the convention that clock names are capitalized and that the inverse of a clock has the same name but is not capitalized. Also, real times are denoted by small letters and clock times by capital letters.

A *system of processes*, denoted (P,N,S), consists of a set of processes, one for each name in P, a nonempty subset N of P called the *nonfaulty* processes, and a nonempty subset S of P called the *self-starting* processes. (We will use P to denote both the set of names and the set of processes, relying on context to distinguish the two.) The nonfaulty processes represent those processes that are required to follow the algorithm. The self-starting processes are intended to model those that will begin executing the algorithm on their own, without first receiving a message. A *system of processes with clocks*, denoted (P,N,S,PH), is a system of processes (P,N,S) together with a set of clocks PH = $\{Ph_p\}$, one for each p in P. Clock Ph_p is called p's *physical clock*. The transition function for p is denoted by τ_p. Throughout this thesis we assume $|P| = n$.

2.4 Message System

We assume that every process can communicate directly with every process, (including itself, for uniformity) at each step. The message system is modelled by a message buffer, which stores each message, together with the real times at which it is sent and delivered. For technical convenience, we do not require that messages be sent before being received. This correctness condition is imposed later.

A state of the message buffer consists of a multiset of tuples, each of the form (p,x,q) or (TIMER,T,p) or (START,p), with associated real times of sending and delivery. The message (x,p) with recipient q is represented by (p,x,q). (TIMER,T,p) indicates a timer set for time T on p's physical clock. (START,p) represents a START message with p as the recipient.

An initial state of the message buffer is a state consisting of some set of START messages. The sending and delivery times are all initialized as ∞.

The behavior of the message buffer is captured as a set of sequences of SEND and RECEIVE operations, each operation with its associated real time. Each operation involves a message tuple. The result of performing each operation is described below.

SEND(u,t): the tuple u is placed in the message buffer with sending time t and delivery time ∞ as long as there is no u entry already in the message buffer with sending time ∞. If there is, then t is made the new sending time of the u entry with the earliest delivery time and sending time ∞.

RECEIVE(u,t): the tuple u is placed in the message buffer with delivery time t and sending time ∞, as long as there is no u entry already in the message buffer with delivery time ∞. If there is, then t is made the new delivery time of the u entry with the earliest sending time and delivery time ∞.

The *message delay* of a non-START message is the delivery time minus the sending time. A positive message delay means the message was sent before it was delivered. A negative message delay means the message was delivered before it was sent. A message delay of $+\infty$ means the message was sent but never delivered, and a message delay of $-\infty$ means the message was delivered, but never sent. (The message delay is not defined for START messages that are never delivered.)

2.5 Histories

In this section we define a history, a construct that models a computation in which nonfaulty processes follow their state-transition functions. Constraints to ensure that the message system behaves correctly will be added in Section 2.8.

Fix a system of processes and clocks $\mathcal{S}$ = (P,N,S,PH).

An *event* for P is of the form receive(m,p), the receipt of message m by process p, where p is in P. A *schedule* for P is a mapping from $\mathbb{R}$ (real times) to finite sequences of events for P such that only a finite number of events occur before any finite time, and for each real time t and process p, all TIMER events for p are ordered after all non-TIMER events for p. The first condition rules out a process taking an infinite number of steps in a finite amount of time, and the second condition allows messages that arrive at the same time as a timer goes off to get in "just under the wire".

In order to discuss how an event affects the system as a whole, we define a *configuration* for P to consist of a state for each process in P and a state for the message buffer. An *initial configuration* for (P,N,S) consists of an initial state for each process and an initial state for the message buffer.

An *action* for P is a triple (F,e,F'), consisting of an event for P and two configurations F and F' for P. F is the *preceding* and F' the *succeeding* configuration for the action.

A *history* for $\mathcal{S}$ is a mapping from real times to sequences of actions for (P,N,S) with the following properties:

- the projection onto the events is a schedule;
- if the sequence of actions is nonempty, then the preceding configuration of the first action is an initial configuration, and the succeeding configuration of each action is the same as the preceding configuration of the following action;
- if an action (F,receive(m,p),F') occurs at real time t, then F = F' except for p's state and the state of the message buffer; moreover, there exist Y in $\mathcal{F}$(X x P) and Z in $\mathcal{F}(\mathbb{R})$ such that the buffer in F' is obtained from the buffer in F by executing the following operations:
  - if m = START, then RECEIVE((START,p),t);
    if m = TIMER, then RECEIVE((TIMER,Ph_p(t),p),t);
    if m = (x,p') for some p', then RECEIVE((p',x,p),t);
  - SEND((p,x,p'),t) for all messages of the form (x,p') in Y;
  - SEND((TIMER,T,p),t) for all T in Z such that T > r (that is, as long as the timer is set for a future time); if T $\leq$ r, then no operation is performed.

Furthermore, if p is in N, then (q,r,m,q',Y,Z) is a step of p, where q is p's state in F, r = $Ph_p(t)$, and q' is p's state in F'.

The first condition merely ensures that only a finite number of occurrences take place by any finite time. The second condition states that the configurations match up correctly. The final condition causes the configurations to change according to the process' transition function, if it is nonfaulty. Since a faulty process need not obey its transition function, it can send any messages and set any timers.

Given $\mathcal{S}$, an initial configuration F, and a schedule s, a history can be constructed inductively by starting with F and applying the transition functions as specified by the events in s to determine the next configuration. We will denote the history so derived by *hist(s,F,$\mathcal{S}$)*.

Define, for each process p and history h, *first-step(h,p)* = min{t: h(t) contains an event for p}. This is the earliest time at which a step is taken by p in h. If p never takes a step, then first-step(h,p) is ∞. Let *first-step(h)* = $\min_{p \in P}$\{first-step(h,p)\}. This is the earliest time at which any process takes a step in h. Similarly, define, for each history h and nonfaulty process p, *last-step(h,p)* = min{t: h(t) contains a configuration in which p is in a final state}. This is the earliest time at which p is a final state. Define *last-step(h)* = $\max_{p \in P}$\{last-step(h,p)\}. This is the earliest time in h after which all nonfaulty processes are in final states. If some p in N never enters a final state in h, then last-step(h,p) and last-step(h) are ∞.

2.6 Chronicles

In order to isolate the steps of an individual process in a history from the real times at which they occur, we define a chronicle.

The *chronicle* of nonfaulty process p in history h is the sequence of tuples of the form $(q_i,r_i,m_i,q_i',Y_i,Z_i)$ which is derived as follows: if the i-th action for p occurs in h(t), then m_i is the message received in that action, q_i is the state of p in the preceding configuration of the action, r_i is p's physical clock reading at real time t, q_i' is the state of p in the succeeding configuration, Y_i is the collection of messages to be sent to the message buffer, and Z_i is the collection of timers to be set. We know that each tuple is a step of p.

Two histories, h for $\mathcal{S}$ = (P,N,S,PH) and h' for $\mathcal{S}'$ = (P,N,S,PH'), are *equivalent* if, for each process p in N, the chronicle of p in h is the same as the chronicle of p in h'.

2.7 Shifting

Given a schedule s, nonfaulty process p, and real number ζ, define a new schedule s' = *shift*(s,p,ζ) to be the same as s except that an event for p appears in s'(t) if and only if the same event appears in s(t + ζ), and the order of events for p is preserved. The result s' can easily be seen to be a schedule also. All events involving p are shifted earlier by ζ if ζ is positive, and shifted later by -ζ if ζ is negative.

A set of clocks PH = $\{Ph_q\}_{q \in P}$ can also be shifted. Let PH' = *shift*(PH,p,ζ) for p in N be the set of clocks defined by PH' = $\{Ph_q'\}_{q \in P}$ where $Ph_q'(t) = Ph_q(t)$ if $q \neq p$, and $Ph_p'(t) = Ph_p(t) + \zeta$. Process p's clock has been shifted forward by ζ, but no other clocks are altered.

Lemma 2-1 states that if a schedule and a set of clocks are shifted by the same amount relative to the same process, then the histories derived from those schedules and sets of clocks starting from the same initial configuration are equivalent.

Lemma 2-1: Let $\mathcal{S}$ = (P,N,S,PH) and $\mathcal{S}'$ = (P,N,S,PH'), where PH' = shift(PH,p,ζ) for some process p and real number ζ. Let s be a schedule for P and s' = shift(s,p,ζ). Let F be an initial configuration for $\mathcal{S}$ and $\mathcal{S}'$. Then the history hist(s,F,$\mathcal{S}$) = h is equivalent to the history hist(s',F,$\mathcal{S}'$) = h'.

Proof: Let q be an arbitrary process in N. It suffices to show that the chronicle of q in h is the same as the chronicle of q in h'.

Case 1: $q \neq p$. We proceed by induction on the elements of the chronicles. Let q's chronicle in h be $(m_i,qc_i,Ph_q(t_i),qn_i,Y_i,Z_i)$ and in h' be $(m_i',qc_i',Ph_q'(t_i'),qn_i',Y_i',Z_i')$. (qc stands for current state, qn for next state.)

Basis: i = 1. Then t_1 = first-step(h,q) and t_1' = first-step(h',q). By construction of h', these real times are the same. Therefore, $m_1 = m_1'$. Since F is the initial configuration in both h and h', $qc_1 = qc_1'$. $Ph_q(t_1) = Ph_q'(t_1')$ since $Ph_q = Ph_q'$ by construction. Finally, $qn_1 = qn_1'$, $Y_1 = Y_1'$, and $Z_1 = Z_1'$ since τ_q is deterministic and the inputs are the same.

Induction: Assume the elements are the same up to i – 1, and show that the i-th elements are the same. Again, $m_i = m_i'$ by construction of h'; $qc_i = qc_i'$ by the induction hypothesis since $qc_i = qn_{i-1} = qn_{i-1}' = qc_i'$; $Ph_q(t_i) = Ph_q'(t_i')$ as before; finally $qn_i = qn_i'$, $Y_i = Y_i'$, and $Z_i = Z_i'$ because τ_q is deterministic.

Case 2: q = p. Again we proceed by induction on the elements of the chronicles. Let p's chronicle in h be $(m_i,qc_i,Ph_p(t_i),qn_i,Y_i,Z_i)$ and in h' be $(m_i',qc_i',Ph_p'(t_i'),qn_i',Y_i',Z_i')$.

First we note that by construction, $t_i = t_i' + \zeta$ for all i.

Basis: i = 1. By construction, $m_1 = m_1'$. Since F is the initial configuration in both h and h', $qc_1 = qc_1'$. $Ph_p(t_1) = Ph_p'(t_1')$ since $Ph_p(t_1) = Ph_p'(t_1-\zeta) = Ph_p'(t_1'+\zeta-\zeta)$. Finally, $qn_1 = qn_1'$, $Y_1 = Y_1'$, and $Z_1 = Z_1'$ since τ_p is deterministic and the inputs are the same.

Induction: assume the elements are the same up to $i - 1$, and show that the i-th elements are the same. $m_i = m_i'$ by construction of h'; $qc_i = qc_i'$ by the induction hypothesis; $Ph_p(t_i) = Ph_p'(t_i')$ by the same argument as in the basis case; and again $qn_i = qn_i'$, $Y_i = Y_i'$, and $Z_i = Z_i'$ since τ_p is deterministic. ∎

The next lemma quantifies the changes to the message delays in a history when its schedule and set of clocks are shifted by the same amount relative to the same process.

Lemma 2-2: Let $\mathcal{S}$ = (P,N,S,PH) and $\mathcal{S}'$ = (P,N,S,PH'), where PH' = shift(PH,p,ζ) for some p in P and real number ζ. Let s be a schedule for P and s' = shift(s,p,ζ). Let F be an initial configuration for $\mathcal{S}$ and $\mathcal{S}'$. Then there is a one-to-one correspondence between the tuples in the message buffer in h = hist(s,F,$\mathcal{S}$) and h' = hist(s',F,$\mathcal{S}'$), and the message delays for corresponding elements will be the same in the two histories (if defined) except for two cases:

1. if the delay for any tuple of the form (p,x,q) is μ in h for any process q $\neq$ p and message value x, then the delay for the corresponding element in h' will be $\mu + \zeta$; and

2. if the delay for any tuple of the form (q,x,p) is μ in h for any process q $\neq$ p and message value x, then the delay for the corresponding element in h' will be $\mu - \zeta$.

Proof: By Lemma 2-1, h and h' are equivalent. Therefore, the chronicles of all the processes are the same. The same messages are sent and received at the same physical clock times in h' and h. Also, the message buffers have the same START elements since the initial configuration is the same for both. Therefore, each element of the message buffer in h has a corresponding one in h' and vice versa.

START messages are still either received at some finite time or not, thus START elements have the same delays in the two histories. Since only p's clock is shifted, the clocks of the other processes will bear the same relationship to real time in h' as in h, causing the delays for messages between processes other than p and the delays of timers for processes other than p to be the same in the two histories. The delays of timers for p will be the same as well, since they are both set and received ζ earlier in h' than in h.

Choose q $\neq$ p.

1. Suppose (p,x,q) is sent at t and received at t' in h. The relationship between s and s' implies that (p,x,q) is sent at $t - \zeta$ and received at t' in h'. Thus the message delay in h' is $t' - (t - \zeta) = \mu + \zeta$.

2. Suppose (q,x,p) is sent at t and received at t' in h. The relationship between s and s' implies that (q,x,p) is sent at t and received at $t' - \zeta$ in h'. Thus the message delay in h' is $t' - \zeta - t = \mu - \zeta$.

∎

2.8 Executions

Now we require correct behavior of the message system. Accordingly, we define an execution to be a history with the necessary properties.

We fix for the remainder of the thesis two nonnegative constants δ and ε with $\delta > \varepsilon$.

An *execution* for $\mathcal{I}$ is a history for $\mathcal{I}$ with four additional properties:

- the initial state of the message buffer consists exactly of a START message for each process in $S \cup (P - N)$, that is, for each self-starting process and each faulty process;
- all START messages for nonfaulty processes are received at some finite time;
- the message delay of any non-TIMER and non-START message is between $\delta - \varepsilon$ and $\delta + \varepsilon$ inclusive; and
- any (TIMER,T,p) element of the message buffer, for any T and p, has finite message delay and is delivered at $Ph_p^{-1}(T)$.

The intent of the first condition is to model the self-starting processes as those processes that begin the algorithm on their own, and to allow the faulty processes to begin their bad behavior at arbitrary times. The second condition states that nonfaulty self-starting processes all receive their START messages. The third condition guarantees that all interprocess messages arrive at their destinations within δ of being sent, subject to an uncertainty of ε. The fourth condition ensures that a timer goes off if and only if it was previously set and that it goes off at the right time.

2.9 Logical Clocks

Each process p has as part of its state a local variable CORR, which provides a correction to its physical clock to yield the local time. During an execution, p's local variable CORR takes on different values. Thus, for a particular execution, it makes sense to define a function $CORR_p(t)$, giving the value of p's variable CORR at time t. For a particular execution, we define the *local time* for p to be the function L_p, which is given by $Ph_p + CORR_p$.

A *logical clock* of p is Ph_p plus the value of $CORR_p$ at some time. Let C^0_p denote the initial logical clock of p, given by Ph_p plus the value of $CORR_p$ in p's initial state. Each time p adjusts its CORR variable, it is, in effect, changing to a new logical clock C^i_p for some i. The local time can be thought of as a piecewise continuous function, each of whose pieces is part of a logical clock.

Chapter Three

Lower Bound

3.1 Introduction

In this chapter, we show a lower bound on how closely clocks can be synchronized, even if the clocks don't drift and no processes are faulty. Since these are strong assumptions, this lower bound also holds for the more realistic case in which clocks do drift and arbitrary faults occur. Just to show that the bound is tight, we present a simple algorithm that synchronizes the clocks as closely as the lower bound.

3.2 Problem Statement

For this chapter alone we make the following assumptions:

1. clocks don't drift, i.e. $dC_p(t)/dt = 1$ for all p and t;
2. all processes are nonfaulty, i.e. N = P. Therefore, we will omit "N" from the notation.

Since the processes have physical clocks which are progressing at the same rate as real time, the only part of the clock synchronization problem which is of interest is the problem of bringing the clocks into synchronization -- once this has been done, synchronization is maintained automatically.

A clock synchronization algorithm (P,S) is *γ,α-correct* if every execution h for (P,S,PH), for any set of clocks PH, satisfies the following three conditions:

1. Termination: All processes eventually enter final states. Thus, last-step(h) is defined.
2. Agreement: $|L_p(t) - L_q(t)| \leq \gamma$ for any processes p and q and time t $\geq$ last-step(h). We say h *synchronizes to within* γ.
3. Validity: For any process p there exist processes q and r such that $C^0_q(t) - \alpha \leq L_p(t) \leq C^0_r(t) + \alpha$ for all times t $\geq$ last-step(h). This ensures that p's new logical clock isn't too much greater (or smaller) than the largest (or smallest) old logical clock would have been at this time. We say h *bounds the adjustment within* α.

We will show that no algorithm can be γ,α-correct for $\gamma < 2\varepsilon(1 - 1/n)$ and any α, where ε is the uncertainty in the message delivery time and n is the number of processes. Then we exhibit a simple algorithm that is $2\varepsilon(1-1/n),\varepsilon$-correct.

3.3 Lower Bound

In this section we show that no algorithm can synchronize n processes' clocks any closer than $2\varepsilon(1-1/n)$.

Theorem 3-1: No clock synchronization algorithm can synchronize a system of n processes to within γ, for any $\gamma < 2\varepsilon(1-1/n)$.

Proof: Fix a system of processes (P,S) that synchronizes to within γ. We will show that $\gamma \geq 2\varepsilon(1-1/n)$.

Let P consist of processes p_1 through p_n. Consider the system $\mathcal{S}_1 = (P,S,PH_1)$. Consider an execution $h_1 = hist(s_1,F,\mathcal{S}_1)$, for some schedule s_1 and initial configuration F, of any clock synchronization algorithm in which all messages from p_j to p_k have delay $\delta - \varepsilon$ if $k > j$, have delay $\delta + \varepsilon$ if $k < j$, and have delay δ if $k = j$.

Consider $n-1$ additional histories, h_2 for system $\mathcal{S}_2$ through h_n for $\mathcal{S}_n$. The systems are constructed inductively by letting $PH_i = shift(PH_{i-1},p_{i-1},2\varepsilon)$ and $\mathcal{S}_i = (P,S,PH_i)$. The histories are constructed inductively by letting $s_i = shift(s_{i-1},p_{i-1},2\varepsilon)$ and $h_i = hist(s_i,F,\mathcal{S}_i)$. Stated informally, the i-th history is obtained from the (i–1)-st history by shifting the schedule and set of clocks by 2ε relative to the (i–1)-st process. Let Ph^i_p be p's physical clock in PH_i.

By Lemma 2-1, all the h_i are equivalent.

Next we show by induction on i that h_i is an execution for $\mathcal{S}_i$, and further, that the delays in h_i for messages from p_j to p_k are $\delta + \varepsilon$ if $j < i$ and $k \geq i$, $\delta - \varepsilon$ if $j \geq i$ and $k < i$, otherwise as in h_1.

Basis: h_1 is an execution and the message delays are as required by hypothesis.

Induction: Assume h_i is an execution with the required message delays, and show that h_{i+1} is also an execution with the required message delays.

- The initial state of the message buffer is the same in h_{i+1} as in h_i, since both use initial configuration F. Thus the initial state is as required.

- The START messages are all received in h_{i+1} as they are in h_i.

- By Lemma 2-2, a message in h_{i+1} from p_i to p_m, $m > i$, will have delay $\delta - \varepsilon + 2\varepsilon = \delta + \varepsilon$; one from p_i to p_m, $m < i$, will have delay $\delta - \varepsilon + 2\varepsilon = \delta + \varepsilon$; one from p_m to p_i, $m > i$, will have delay $\delta + \varepsilon - 2\varepsilon = \delta - \varepsilon$; and one from p_m to p_i, $m < i$, will have delay $\delta + \varepsilon - 2\varepsilon = \delta - \varepsilon$. The others stay the same. Thus the delays are within the correct range.

- Now we need to show that timers are handled properly in h_{i+1}. Lemma 2-2 implies that the message delays are the same in h_{i+1} as in h_i, thus they are finite. For all processes except p_i, the timers arrive at the same real times and the same clock times in h_{i+1} as in h_i, and thus they arrive at the proper times in h_{i+1}. Consider a timer set by p_i for T that arrives at $T = Ph^i_{p_i}(t)$ in h_i. In h_{i+1} it arrives at $t + 2\varepsilon$. However, since $Ph^{i+1}_{p_i}(t+2\varepsilon) = Ph^i_{p_i}(t) = T$, the timer arrives at the proper time in h_{i+1}.

Therefore, h_i is an execution for $\mathcal{I}_i$.

Since h_1 was correct, it terminated; therefore, h_i also terminates. Let $t_f = \max_{i=1..n}\{\text{last-step}(h_i)\}$. In execution h_1, the algorithm synchronizes all the processes' clocks to values v_1 through v_n at time t_f, and all the values are within γ. In particular,

$$v_n \leq v_1 + \gamma.$$

Since h_i is equivalent to h_{i-1}, the correction variable for any process p will be the same in both executions at time t_f. The value of p_{i-1}'s logical clock at t_f will be $v_{i-1} + 2\varepsilon$ and the value of p_i's logical clock at t_f will be v_i by the way PH_i is defined. Since these values are within γ, we have

$$v_{i-1} \leq v_i + \gamma - 2\varepsilon.$$

Putting together this chain of inequalities, we have

$$v_n \leq v_1 + \gamma \leq \ldots \leq v_i + (i-1)(\gamma-2\varepsilon) + \gamma \leq \ldots \leq v_n + (n-1)(\gamma-2\varepsilon) + \gamma.$$

Therefore, $v_n \leq v_n + (n-1)(\gamma - 2\varepsilon) + \gamma$, and so $0 \leq (n-1)\gamma - (n-1)2\varepsilon + \gamma$. In order for this inequality to hold, it must be the case that $\gamma \geq 2\varepsilon(1 - 1/n)$. ∎

3.4 Upper Bound

In this section we show that the $2\varepsilon(1 - 1/n)$ lower bound is tight, by exhibiting a simple algorithm which synchronizes the clocks to within this amount.

3.4.1 Algorithm

There is an extremely simple algorithm that achieves the closest possible synchronization. As soon as each process p receives a message, it sends its local time in a message to the remaining processes and waits to receive a similar message from every other process. Immediately upon receiving such a message, say from q, p estimates q's current local time by adding δ to the value received. Then p computes the difference between its estimate of q's local time and its own current local time. After receiving local times from all the other processes, p takes the average of the estimated differences (including 0 for the difference between p and itself) and adds this average to its correction variable. Note that in contrast to many other agreement algorithms, in this one each process treats itself non-uniformly with the others.

Since it is obviously impractical to write algorithms in terms of transition functions, we have employed a clean, simple notation for describing interrupt-driven algorithms. To translate this notation into the basic model, we first assume that the state of a process consists of values for all the local variables, together with a location counter which indicates the next *beginstep* statement to be executed. The initial state of a process consists of the indicated initial values for all the local variables, and the location counter positioned at the first beginstep statement of the program.

The transition function takes as inputs a state of the process, a message, and a physical time, and must return a new state and a collection of messages to send and timers to set. This is done as follows. The beginstep statement is extracted from the given state. The local variables are initialized at the values given in the state. The parameter u is set equal to the message. The variable NOW is initialized at the given physical time + CORR. The program is then run from the given beginstep statement, just until it reaches an *endstep* statement. (If it never reaches an endstep statement, the transition function takes on a default value.) The next beginstep after that endstep, together with the new values for all the local variables resulting from running the program, comprise the new state. The messages sent are all those which are sent during the running of the program, and similarly for the timers.

There is a *set-timer* statement, which takes an argument U representing a logical time. The corresponding physical time, U – CORR, is the physical time described by the transition function. (This statement is not used in this algorithm but will be used later in the thesis.)

We will use the shorthand NOW to stand for the current logical clock time and ME for the id of the process running the code.

For this algorithm, initial states are those in which the location counter is at the beginning of the code, local variables CORR and V have arbitrary values, and local variables SUM and RESPONSES have value 0. Final states are those in which the location counter is at the end of the code.

The code is in Figure 3-1.

We will show that any execution h of Algorithm 3-1 is γ,α-correct, where $\gamma = 2\varepsilon(1 - 1/n)$ and $\alpha = \varepsilon$. Thus, Algorithm 3-1 synchronizes the clocks to within $2\varepsilon(1 - 1/n)$, showing that the lower bound is tight. The upper bound isn't as unintuitive as it might look at first glance; it can be

```
beginstep(u)
send(NOW) to all q ≠ ME

do forever
     if u = (v,q) for some message value v and process q then
          V := v + δ - NOW
          SUM := SUM + V
          RESPONSES := RESPONSES + 1
          endif
     if RESPONSES = n - 1 then exit endif
     endstep
     beginstep(u)
     enddo

CORR := CORR + SUM/n
endstep
```

Figure 3-1: Algorithm 3-1, Synchronizing to within the Lower Bound

rewritten as $(2\varepsilon + (n-2)2\varepsilon)/n$, the average of the discrepancies in the estimated differences. The estimated differences of two processes for each other can differ by at most ε apiece (giving the 2ε term), and their estimated differences for the other $n-2$ processes can differ by up to 2ε apiece (giving the $(n-2)2\varepsilon$ term). Then the estimated differences are averaged, so the sum is divided by n. A more careful analysis is given below.

3.4.2 Preliminary Lemmas

The next two results follow easily from the assumption that clocks don't drift.

Lemma 3-2: For any p and $i \geq 0$, $C^i_p(t') - C^i_p(t) = t' - t$.

Proof: Immediate since the slope of C^i_p is 1. ▮

Lemma 3-3: For any p and q, $i \geq 0$, and times t and t', $C^i_p(t') - C^i_q(t') = C^i_p(t) - C^i_q(t)$.

Proof: $C^i_p(t') - C^i_p(t) = t' - t = C^i_q(t') - C^i_q(t)$ by two applications of Lemma 3-2. The result follows. ▮

Now we can define the *initial difference* between two processes' clocks in execution h. Define Δ_{pq} to be $C^0_p(t) - C^0_q(t)$. That is, Δ_{pq} is the difference in local times before either of the processes has changed its correction variable. Since there is no drift in the clock rates, any time will give the same value.

Lemma 3-4: For any execution h, and processes p and q, $\Delta_{pq} = -\Delta_{qp}$.

Proof: Immediate from the definition of Δ. ▮

Lemma 3-5: For any execution h, and processes p, q, and r, $\Delta_{pq} = \Delta_{pr} + \Delta_{rq}$.

Proof: Immediate from the definition of Δ. ▮

3.4.3 Agreement

For $q \neq p$, let V_{qp} be the value of variable V in the code when q's message is being handled by p. $V_{qp} = L_q(t) + \delta - L_p(t')$, where local time $L_q(t)$ was sent by q at real time t and received by p at real time t'. Let $V_{pp} = 0$. We will denote SUM/n, p's addition to its correction variable, by A_p.

First we relate the estimate V_{qp} to the actual value Δ_{qp}.

Lemma 3-6: $|V_{qp} - \Delta_{qp}| \leq \varepsilon$.

Proof: Suppose at real time t, q sent the value $L_q(t)$, which was received by p at real time t'. Then

$$|V_{qp} - \Delta_{qp}| = |L_q(t) + \delta - L_p(t') - \Delta_{qp}| = |C^0_q(t) + \delta - C^0_p(t') - \Delta_{qp}|$$

$= |C^0_p(t) + \Delta_{qp} + \delta - C^0_p(t') - \Delta_{qp}|$, by definition of Δ_{qp}

$= |C^0_p(t) - C^0_p(t') + \delta|$

$= |t - t' + \delta|$, by Lemma 3-2

$= |\delta - (t' - t)|$

$\leq |\delta - (\delta - \varepsilon)|$, since $\delta - \varepsilon$ is the smallest message delay

$= \varepsilon$. ∎

Here is the main result.

Theorem 3-7: (Agreement) Algorithm 3-1 guarantees clock synchronization to within $2\varepsilon(1 - 1/n)$.

Proof: We must show that for any execution h, any two processes p and q, and all times t after last-step(h),

$$|L_p(t) - L_q(t)| \leq 2\varepsilon - 2\varepsilon/n.$$

Without loss of generality, assume $p = p_1$ and $q = p_2$, so that the remaining processes are p_3 through p_n. By the way the algorithm works,

$$|L_p(t) - L_q(t)| = |(C^0_p(t) + A_p) - (C^0_q(t) + A_q)| = |\Delta_{pq} + A_p - A_q|.$$

We know by definition of A_p and A_q that

$A_p = (1/n)(V_{pp} + V_{qp} + \Sigma_{i=3..n} V_{p_i p})$ and

$A_q = (1/n)(V_{pq} + V_{qq} + \Sigma_{i=3..n} V_{p_i q})$.

Substituting these values and noting that $V_{pp} = V_{qq} = 0$, we get

$$|L_p(t) - L_q(t)| = |\Delta_{pq} + (1/n)(V_{qp} + \Sigma_{i=3..n} V_{p_i p} - V_{pq} - \Sigma_{i=3..n} V_{p_i q})|$$

$= (1/n)|n\Delta_{pq} + V_{qp} + \Sigma_{i=3..n}V_{p_ip} - V_{pq} - \Sigma_{i=3..n}V_{p_iq}|$

$= (1/n)|(\Delta_{pq} + V_{qp}) + (\Delta_{pq} - V_{pq}) + \Sigma_{i=3..n}(\Delta_{pq} + V_{p_ip} - V_{p_iq})|$

$\leq (1/n)(|\Delta_{pq} + V_{qp}| + |\Delta_{pq} - V_{pq}| + \Sigma_{i=3..n}|\Delta_{pq} + V_{p_ip} - V_{p_iq}|)$

$\leq (1/n)(\varepsilon + \varepsilon + \Sigma_{i=3..n}|\Delta_{pq} + V_{p_ip} - V_{p_iq}|)$, by Lemmas 3-6 and 3-4

$= (1/n)(2\varepsilon + \Sigma_{i=3..n}|\Delta_{pp_i} + \Delta_{p_iq} + V_{p_ip} - V_{p_iq}|)$, by Lemma 3-5

$= (1/n)(2\varepsilon + \Sigma_{i=3..n}|(V_{p_ip} - \Delta_{p_ip}) - (V_{p_iq} - \Delta_{p_iq})|)$, by Lemma 3-4

$\leq (1/n)(2\varepsilon + \Sigma_{i=3..n}|V_{p_ip} - \Delta_{p_ip}| + \Sigma_{i=3..n}|-(V_{p_iq} - \Delta_{p_iq})|)$

$\leq (1/n)(2\varepsilon + \Sigma_{i=3..n}\varepsilon + \Sigma_{i=3..n}\varepsilon)$, by Lemma 3-6

$\leq (1/n)(2\varepsilon + (n-2)2\varepsilon)$

$= 2\varepsilon(1 - 1/n)$. ∎

3.4.4 Validity

The validity result states that each new logical clock is within ε of what one of the initial logical clocks would have been.

Theorem 3-8: (Validity) Algorithm 3-1 bounds the adjustment within ε.

Proof: By definition, the amount to be added to $CORR_p$ is $A_p = (1/n)\ \Sigma_{q \in P}V_{qp}$. Then $\min_{q \in P}V_{qp} \leq A_p \leq \max_{r \in P}V_{rp}$. Let q be the process with the minimum V_{qp}. Let r be the process with the maximum V_{rp}. Then,

$V_{qp} \leq A_p \leq V_{rp}$.

By applying Lemma 3-6 to each end of this inequality, we get

$\Delta_{qp} - \varepsilon \leq V_{qp} \leq A_p \leq V_{rp} \leq \Delta_{rp} + \varepsilon$.

Adding p's initial clock value $C^0_p(t)$ for $t \geq t_f$, we get

$C^0_p(t) + \Delta_{qp} - \varepsilon \leq C^0_p(t) + A_p \leq C^0_p(t) + \Delta_{rp} + \varepsilon$,

which together with the definition of Δ implies

$C^0_q(t) - \varepsilon \leq L_p(t) \leq C^0_r(t) + \varepsilon$. ∎

Chapter Four

Maintenance Algorithm

4.1 Introduction

This chapter consists of an algorithm to keep synchronized clocks that are close together initially, and an analysis of its performance concerning how closely the clocks are synchronized and how close the clocks stay to real time. The algorithm handles clock drift and arbitrary process faults. The algorithm requires the clocks to be initially close together and less than one third of the processes to be faulty. (Dolev, Halpern and Strong [2] show that it is impossible without authentication to synchronize clocks unless more than two thirds of the processes are nonfaulty.)

This algorithm runs in rounds, resynchronizing periodically to correct for clock drift, and using a fault-tolerant averaging function based on those in [1] to calculate an adjustment. The size of the adjustment is independent of the number of faulty processes. At each round, n^2 messages are required, where n is the total number of processes. The closeness of synchronization achieved depends only on the initial closeness of synchronization, the message delivery time and its uncertainty, and the drift rate. We give explicit bounds on how the difference between the clock values and real time grows as time proceeds. The algorithm can be easily adapted to include reintegration of repaired processes as described in Section 4.8.

4.2 Problem Statement

We are now considering the situation in which clocks can drift slightly and some proportion of the processes can be faulty. Therefore, the statement of the problem differs from that in Chapter 3.

For a very small constant $\rho > 0$, we define a clock C to be *ρ-bounded* provided that for all t

$$1 - \rho \leq 1/(1 + \rho) \leq dC(t)/dt \leq 1 + \rho \leq 1/(1 - \rho).$$

We make the following assumptions:

1. All clocks are ρ-bounded, including those of faulty processes, i.e., the amount by which a clock's rate is faster of slower than real time is at most ρ. (Since faulty processes are permitted to take arbitrary steps, faulty clocks would not increase their

power to affect the behavior of nonfaulty processes.)

2. There are at most f faulty processes, for a fixed constant f, and the total number of processes in the system, n, is at least 3f + 1.

3. A START message arrives at each process p at time T^0 on its initial logical clock C^0_p, and t^0_p is the real time when this occurs. Furthermore, the initial logical clocks are closely synchronized, i.e., $|c^0_p(T^0) - c^0_q(T^0)| \leq \beta$, for some fixed β and all nonfaulty p and q.

We let $tmax^0 = max_{p\ nonfaulty}\{t^0_p\}$ and analogously for $tmin^0$.

The object is to design an algorithm for which every execution in which the assumptions above hold satisfies the following two properties.

1. γ-Agreement: $|L_p(t) - L_q(t)| \leq \gamma$, for all $t \geq tmin^0$ and all nonfaulty p, q.

2. $(\alpha_1,\alpha_2,\alpha_3)$-Validity: $\alpha_1(t - tmax^0) + T^0 - \alpha_3 \leq L_p(t) \leq \alpha_2(t - tmin^0) + T^0 + \alpha_3$, for all $t \geq t^0_p$ and all nonfaulty p.

The Agreement property means that all the nonfaulty processes are synchronized to within γ. The Validity property means that the local time of a nonfaulty process increases in some relation to real time. We would, of course, like to minimize α_1, α_2, α_3, and γ.

4.3 Properties of Clocks

We give several straightforward lemmas about the behavior of (ρ- bounded) clocks.

Lemma 4-1: Let C be any clock.

(a) If $t_1 \leq t_2$, then

$(1 - \rho)(t_2 - t_1) \leq (t_2 - t_1)/(1 + \rho) \leq C(t_2) - C(t_1) \leq (1 + \rho)(t_2 - t_1) \leq (t_2 - t_1)/(1 - \rho)$.

(b) If $T_1 \leq T_2$, then

$(1-\rho)(T_2 - T_1) \leq (T_2 - T_1)/(1 + \rho) \leq c(T_2) - c(T_1) \leq (1 + \rho)(T_2 - T_1) \leq (T_2 - T_1)/(1-\rho)$.

Proof: Straightforward. ∎

Lemma 4-2: Let C and D be clocks.

(a) If $dC(t)/dt = 1$ and $T_1 \leq T_2$, then

$|(c(T_2) - d(T_2)) - (c(T_1) - d(T_1))| = |(c(T_2) - c(T_1)) - (d(T_2) - d(T_1))| \leq \rho(T_2 - T_1)$.

(b) If $T_1 \leq T_2$, then

$|(c(T_2) - d(T_2)) - (c(T_1) - d(T_1))| = |(c(T_2) - c(T_1)) - (d(T_2) - d(T_1))| \leq 2\rho(T_2 - T_1)$.

(c) If $dC(t)/dt = 1$ and $t_1 \leq t_2$, then

$|(C(t_2) - D(t_2)) - (C(t_1) - D(t_1))| = |(C(t_2) - C(t_1)) - (D(t_2) - D(t_1))| \leq \rho(t_2 - t_1)$.

(d) If $t_1 \leq t_2$, then

$|(C(t_2) - D(t_2)) - (C(t_1) - D(t_1))| = |(C(t_2) - C(t_1)) - (D(t_2) - D(t_1))| \leq 2\rho(t_2 - t_1)$.

Proof: Straightforward using Lemma 4-1. ∎

Lemma 4-3: Let C and D be clocks, $T_1 \leq T_2$. Assume $|c(T) - d(T)| \leq \alpha$ for all T, $T_1 \leq T \leq T_2$. Let $t_1 = \min\{c(T_1),d(T_1)\}$ and $t_2 = \max\{c(T_2),d(T_2)\}$.

Then $|C(t) - D(t)| \leq (1 + \rho)\alpha$ for all t, $t_1 \leq t \leq t_2$.

Proof: There are four cases, which can easily be shown to be exhaustive.

Case 1: $c(T_1) \leq t \leq c(T_2)$.

Let $T_3 = C(t)$, so that $T_1 \leq T_3 \leq T_2$. By hypothesis, $|c(T_3) - d(T_3)| \leq \alpha$. Then $|T_3 - D(t)| \leq (1 + \rho)\alpha$, by Lemma 4-1.

Case 2: $d(T_1) \leq t \leq d(T_2)$. This case is analogous to the first.

Case 3: $c(T_2) < t < d(T_1)$.

Then $c(T_1) < t < d(T_1)$. So $C(t) > D(t)$, and thus

$|C(t) - D(t)| = C(t) - D(t) = (C(t) - T_1) + (T_1 - D(t))$

$\leq (1 + \rho)(t - c(T_1)) + (1 + \rho)(d(T_1) - t)$, by Lemma 4-1,

$= (1 + \rho)(d(T_1) - c(T_1)) \leq (1 + \rho)\alpha$.

Case 4: $d(T_2) < t < c(T_1)$. This case is analogous to the third. ∎

4.4 The Algorithm

4.4.1 General Description

The algorithm executes in a series of rounds, the i-th round for a process triggered by its logical clock reaching some value T^i. (It will be shown that the logical clocks reach this value within real time β of each other.) When any process p's logical clock reaches T^i, p broadcasts a T^i message. Meanwhile, p collects T^i messages from as many processes as it can, within a particular bounded amount of time, measured on its logical clock. The bounded amount of time is of length $(1 + \rho)(\beta$

$+ \delta + \varepsilon$), and is chosen to be just large enough to ensure that T^i messages are received from all nonfaulty processes. After waiting this amount of time, p averages the arrival times of all the T^i messages received, using a particular fault-tolerant averaging function. The resulting average is used to calculate an adjustment to p's correction variable, thereby switching p to a new logical clock.

The process p then waits until its new clock reaches time $T^{i+1} = T^i + P$, and repeats the procedure. P, then, is the length of a round in local time.

The fault-tolerant averaging function is derived from those used in [1] for reaching approximate agreement. The function is designed to be immune to some fixed maximum number, f, of faults. It first throws out the f highest and f lowest values, and then applies some ordinary averaging function to the remaining values. In this paper, we choose the midpoint of the range of the remaining values, to be specific.

4.4.2 Code for an Arbitrary Process

Global constants: ρ, β, δ, ε, and P, as defined above.

Local variables:

- CORR, initially arbitrary; correction variable which corrects physical time to logical time.
- ARR[q], initially arbitrary; array containing the arrival times of the most recent messages, one entry for each process q.
- T, initially undefined; local time at which the process next intends to send a message.

Conventions:

- NOW stands for the current logical clock time (i.e., the physical clock reading + CORR). NOW is assumed to be set at the beginning of a step, and cannot be assigned to.
- REDUCE, applied to an array, returns the multiset consisting of the elements of the array, with the f highest and f lowest elements removed.
- MID, applied to a multiset of reals numbers, returns the midpoint of the set of values in the multiset.

The code is in Figure 4-1.

```
beginstep(u)
do forever

/* in case T^i messages are received before this process reaches T^i */

     while u = (m,q) for some message m and process q do
          ARR[q] := NOW
          endstep
          beginstep(u)
          endwhile

/* fall out of the loop when u = START or TIMER; begin round */

     T := NOW
     broadcast(T)
     set-timer(T + (1 + ρ)(β + δ + ε))

     while u = (m,q) for some message m and process q do
          ARR[q] := NOW
          endstep
          beginstep(u)
          endwhile

/* fall out of the loop when u = TIMER; end round */

     AV := mid(reduce(ARR))
     ADJ := T + δ - AV
     CORR := CORR + ADJ
     set-timer(T + P)
     endstep
     beginstep(u)
     enddo
```

Figure 4-1: Algorithm 4-1, Maintaining Synchronization

4.5 Inductive Analysis

Although the algorithm is fairly simple, its analysis is surprisingly complicated and requires a long series of lemmas.

4.5.1 Bounds on the Parameters

We assume that the parameters ρ, δ, and ε are fixed, but that we have some freedom in our choice of P and β, subject to the reasonableness of our assumption that the clocks are initially synchronized to within β. We would like β to be as small as possible, to keep the clocks as closely synchronized as we can. However, the smaller β is, the smaller P must be (i.e., the more frequently we must synchronize).

There is also a lower bound on P. In order for the algorithm to work correctly, we need to have P sufficiently large to ensure the following.

(1) After a nonfaulty process p resets its clock, the local time at which p schedules its next broadcast is greater than the local time on the new clock, at the moment of reset.

(2) A message sent by a nonfaulty process q for a round arrives at a nonfaulty process p after p has already set its clock for that round.

Sufficient bounds on P turn out to be:

$P > 2(1+\rho)(\beta+\varepsilon) + (1+\rho)\max\{\delta, \beta+\varepsilon\} + \rho\delta$, and

$P \leq \beta/4\rho - \varepsilon/\rho - \rho(\beta+\delta+\varepsilon) - 2\beta - \delta - 2\varepsilon$.

A required lower bound on β is $\beta \geq 4\varepsilon + 4\rho(3\beta+\delta+3\varepsilon) + 8\rho^2(\beta+\delta+\varepsilon)$.

Any combination of P and β which satisfies these inequalities will work in our algorithm. If P is regarded as fixed, then β, the closeness of synchronization along the real time axis, is roughly $4\varepsilon + 4\rho P$. This value is obtained by solving the upper bound on P for β and neglecting terms of order ρ.

4.5.2 Notation

Let $T^i = T^0 + iP$ and $U^i = T^i + (1+\rho)(\beta+\delta+\varepsilon)$, for all $i \geq 0$.

For each i, every process p broadcasts T^i at its logical clock time T^i (real time t^i_p) and sets a timer to go off when its logical clock reaches U^i. When the logical clock reaches U^i (at real time u^i_p), the process resets its CORR variable, thereby switching to a new logical clock, denoted C^{i+1}_p. Also at real time u^i_p, the process sets a timer for the time on its physical clock when the new logical clock C^{i+1}_p reaches T^{i+1}. It is at least theoretically possible that this new timer might be set for a time on the physical clock which has already passed. If the timer is never set in the past, the process moves through an infinite sequence of clocks C^0_p, C^1_p, etc, where C^0_p is in force in the interval of real time $(-\infty, u^0_p)$, and each C^i_p, $i \geq 1$, is in force in the interval of real time $[u^{i-1}_p, u^i_p)$. If, however, the timer is set in the past at some u^i_p, then no further timers arrive after that real time, and no further resynchronizations occur. That is, C^{i+1}_p stays in force forever, and u^j_p and t^j_p are undefined for $j \geq i+1$.

Let $tmin^i$ denote $\min_{p\ nonfaulty}\{t^i_p\}$, and analogously for $tmax^i$, $umin^i$ and $umax^i$.

For p and q nonfaulty, let $ARR^i_p(q)$ denote the time of arrival of a T^i message from q to p, sent at q's clock time T^i, where the arrival time is measured on p's local clock C^i_p. (We will prove that C^i_p has actually been set by the time this message arrives.) Let AV^i_p denote the value of AV calculated by p using the ARR^i_p values, and let ADJ^i_p denote the corresponding value of ADJ calculated by p. Thus, $C^{i+1}_p = C^i_p + ADJ^i_p$.

This section is devoted to proving the following three statements for all $i \geq 0$:

(1) The real time t^i_p is defined for all nonfaulty p. (That is, timers are set in the future.)

(2) $|t^i_p - t^i_q| \leq \beta$, for all nonfaulty p and q. (That is, the separation of clocks is bounded by β.)

(3) $t^i_p + \delta - \varepsilon > u^{i-1}_q$, for all nonfaulty p and q, and $i \geq 1$. (That is, messages arrive after the appropriate clocks have been set.)

The proof is by induction. For $i = 0$, (1) and (2) are true by assumption and (3) is vacuously true.

Throughout the rest of this section, we assume (1), (2), and (3) hold for i. We show (1), (2), and (3) for $i + 1$ after bounding the size of the adjustment at each round.

4.5.3 Bounding the Adjustment

In this subsection, we prove several lemmas leading up to a bound on the amount of adjustment made by a nonfaulty process to its clock, at each time of resynchronization.

Lemma 4-4: Let p and q be nonfaulty.

(a) $ARR^i_p(q) \leq T^i + (1 + \rho)(\beta + \delta + \varepsilon)$.

(b) If $\delta - \varepsilon \geq \beta$, then $ARR^i_p(q) \geq T^i + (1 - \rho)(\delta - \varepsilon - \beta)$.

(c) If $\delta - \varepsilon \leq \beta$, then $ARR^i_p(q) \geq T^i - (1 + \rho)(\beta - \delta + \varepsilon)$.

Proof: Straightforward using Lemma 4-1. ∎

Lemma 4-5: Let p be nonfaulty. Then there exist nonfaulty q and r with

$ARR^i_p(q) \leq AV^i_p \leq ARR^i_p(r)$.

Proof: By throwing out the f highest and f lowest values, the process ensures that the remaining values are in the range of the nonfaulty processes' values. ∎

We are now able to bound the adjustment.

Lemma 4-6: Let p be nonfaulty. Then $|ADJ^i_p| \leq (1 + \rho)(\beta + \varepsilon) + \rho\delta$.
Proof: $ADJ^i_p = T^i + \delta - AV^i_p$.

Thus, for some nonfaulty q and r, Lemma 4-5 implies that

$T^i + \delta - ARR^i_p(q) \leq ADJ^i_p \leq T^i + \delta - ARR^i_p(r)$.

Then Lemma 4-4 implies that:

(a) $ADJ^i_p \geq T^i + \delta - (T^i + (1 + \rho)(\beta + \delta + \varepsilon)) = -(1 + \rho)(\beta + \varepsilon) - \rho\delta$.

(b) If $\delta - \varepsilon \geq \beta$, then $ADJ^i_p \leq T^i + \delta - (T^i + (1 - \rho)(\delta - \varepsilon - \beta)) = (1 - \rho)(\beta + \varepsilon) + \rho\delta$.

(c) If $\delta - \varepsilon \leq \beta$, then $ADJ^i_p \leq T^i + \delta - (T^i - (1 + \rho)(\beta - \delta + \varepsilon)) = (1 + \rho)(\beta + \varepsilon) - \rho\delta$.

The conclusion is immediate. ∎

4.5.4 Timers Are Set in the Future

Earlier, we gave a lower bound on P and described two conditions which that bound was supposed to guarantee (that timers are set in the future and that messages arrive after the appropriate clocks have been set). In this subsection, we show that the given bound on P is sufficient to guarantee that the first of these two conditions holds.

Lemma 4-7: Let p be nonfaulty. Then $U^i + ADJ^i_p < T^{i+1}$.
Proof: $U^i + ADJ^i_p \leq U^i + (1 + \rho)(\beta + \varepsilon) + \rho\delta$, by Lemma 4-6

$= U^i + (2(1 + \rho)(\beta + \varepsilon) + (1 + \rho)\delta + \rho\delta) - (1 + \rho)(\beta + \delta + \varepsilon)$

$< U^i + P - (1 + \rho)(\beta + \delta + \varepsilon)$, by the assumed lower bound on P

$= T^{i+1}$. ∎

This lemma implies that timers are set in the future and that t^{i+1}_p is defined, the first of the three inductive properties which we must verify.

4.5.5 Bounding the Separation of Clocks

Next, we prove several lemmas which lead to bounds on the distance between the new clocks of nonfaulty processes. The first lemma gives an upper bound on the error in a process' estimate of the difference in real time between its own clock and another nonfaulty process' clock reaching T^i.

Lemma 4-8: Let p, q and r be nonfaulty. Then

$|(ARR^i_p(q) - (T^i + \delta)) - (c^i_q(T^i) - c^i_p(T^i))| \leq \varepsilon + \rho(\beta + \delta + \varepsilon)$.

Proof: Let a be the real time of arrival of q's message at process p. Then a is at most $c^i_q(T^i) + \delta + \varepsilon$. Define a new auxiliary clock, D, with rate exactly equal to 1, and such that $D(a) = C^i_p(a)$. Thus, $ARR^i_p(q) = D(a)$. So the expression we want to bound is at most equal to:

$$|(D(a) - (T^i + \delta)) - (c^i_q(T^i) - d(T^i))| + |c^i_p(T^i) - d(T^i)|.$$

First we demonstrate that the first of these two terms is at most ε.

$|D(a) - (T^i + \delta) - c^i_q(T^i) + d(T^i)|$

$= |a - d(T^i + \delta) - c^i_q(T^i) + d(T^i)|$, since D has rate 1

$= |a - c^i_q(T^i) + T^i - (T^i + \delta)|$

$\leq |c^i_q(T^i) + \delta + \varepsilon - c^i_q(T^i) - \delta|$

$= \varepsilon$.

Next we show that the second term, $|c^i_p(T^i) - d(T^i)|$, is at most $\rho(\beta + \delta + \varepsilon)$.

Case 1: $c^i_p(T^i) \leq a$. So p reaches T^i before q's message arrives.

Let $\gamma = a - c^i_p(T^i)$. Then $\gamma \leq \beta + \delta + \varepsilon$.

Subcase 1a: $d(T^i) \geq c^i_p(T^i)$. So C_p has rate slower than real time.

Then $d(T^i) - c^i_p(T^i)$ is largest when C_p goes at the slowest possible rate, $1/(1 + \rho)$. In this case, $d(T^i) - c^i_p(T^i) = \gamma - (a - d(T^i))$, where $a - d(T^i) = \gamma/(1 + \rho)$. Thus, $d(T^i) - c^i_p(T^i) = \gamma(1 - 1/(1 + \rho)) = \gamma\rho/(1 + \rho) \leq \gamma\rho \leq \rho(\beta + \delta + \varepsilon)$.

Subcase 1b: $d(T^i) \leq c^i_p(T^i)$. So C_p has rate faster than real time.

Then $c^i_p(T^i) - d(T^i)$ is largest when C_p goes at the fastest possible rate, $1 + \rho$. Then $c^i_p(T^i) - d(T^i) = \gamma(1 + \rho) - \gamma = \gamma\rho \leq \rho(\beta + \delta + \varepsilon)$.

Case 2: $c^i_p(T^i) \geq a$. So p reaches T^i after q's message arrives.

Let $\gamma = c^i_p(T^i) - a$. Then $\gamma \leq \beta - \delta + \varepsilon$.

Subcase 2a: $d(T^i) \geq c^i_p(T^i)$. So C_p has rate faster than real time.

An argument similar to that for case 1b shows that $d(T^i) - c^i_p(T^i) \leq \gamma\rho \leq \rho(\beta - \delta + \varepsilon)$, which suffices.

Subcase 2b: $d(T^i) \leq c^i_p(T^i)$. So C_p has rate slower than real time.

An argument similar to that for case 1a shows that $c^i_p(T^i) - d(T^i) \leq \gamma\rho \leq \rho(\beta - \delta + \varepsilon)$, which suffices. ∎

In order to prove the next lemma, we use some results about multisets, which are presented in the Appendix. This is a key lemma because the distance between the clocks is reduced from β to $\beta/2$, roughly. The halving is due to the properties of the fault-tolerant averaging function used in the algorithm. Consequently, the averaging function can be considered the heart of the algorithm.

Lemma 4-9: Let p and q be nonfaulty. Then

$|(c^i_p(T^i) - c^i_q(T^i)) - (ADJ^i_p - ADJ^i_q)| \leq \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon)$.

Proof: We define multisets U, V, and W, and show they satisfy the hypotheses of Lemma A-4. Let

$U = c^i_p(T^i) - (T^i + \delta) + ARR^i_p$,

$V = c^i_q(T^i) - (T^i + \delta) + ARR^i_q$, and

$W = \{c^i_r(T^i)\text{: r is nonfaulty}\}$.

U and V have size n and W has size $n - f$.

Let $x = \varepsilon + \rho(\beta + \delta + \varepsilon)$.

Define an injection from W to U as follows. Map each element $c^i_r(T^i)$ in W to $c^i_p(T^i) - (T^i + \delta) + ARR^i_p(r)$ in U. Since Lemma 4-8 implies that $|(ARR^i_p(r) - (T^i + \delta)) - (c^i_r(T^i) - c^i_p(T^i))| \leq \varepsilon + \rho(\beta + \delta + \varepsilon)$ for all the elements of W, $d_x(W,U) = 0$. Similarly, $d_x(W,V) = 0$.

Since any two nonfaulty processes reach T^i within β real time of each other, diam(W) $= \beta$.

By Lemma A-4, $|mid(reduce(U)) - mid(reduce(V))| \leq \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon)$.

Since $mid(reduce(U)) = mid(reduce(c^i_p(T^i) - (T^i + \delta) + ARR^i_p)) = c^i_p(T^i) - ADJ^i_p$, and similarly $mid(reduce(V)) = c^i_q(T^i) - ADJ^i_q$, the result follows. ∎

The next lemma is analogous to the previous one, except that it involves U^i instead of T^i.

Lemma 4-10: Let p and q be nonfaulty. Then

$|(c^i_p(U^i) - c^i_q(U^i)) - (ADJ^i_p - ADJ^i_q)| \leq \beta/2 + 2\varepsilon + 2\rho(2 + \rho)(\beta + \delta + \varepsilon)$.

Proof: The given expression is

$\leq |(c^i_p(T^i) - c^i_q(T^i)) - (ADJ^i_p - ADJ^i_q)| + |(c^i_p(U^i) - c^i_q(U^i)) - (c^i_p(T^i) - c^i_q(T^i))|$

$\leq \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon) + 2\rho(1 + \rho)(\beta + \delta + \varepsilon)$, by Lemmas 4-9 and 4-2.

This reduces to the claimed expression. ▮

Next we bound the distance in real time between two nonfaulty processes switching to their new clocks. It is crucial that the distance between the new clocks reaching U^i be less than β in order to accommodate their relative drift during the interval between U^i and T^{i+1}.

Lemma 4-11: Let p, q be nonfaulty. Then

$|c^{i+1}_p(U^i) - c^{i+1}_q(U^i)| \leq \beta/2 + 2\varepsilon + 2\rho(3\beta + 2\delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon)$.

Proof: We define idealized clocks, D_p and D_q, as follows. Both have rate exactly 1. Also, $D_p(u^i_p) = C^{i+1}_p(u^i_p) = U^i + ADJ^i_p$, and similarly for q. Then

$|c^{i+1}_p(U^i) - c^{i+1}_q(U^i)| \leq |c^{i+1}_p(U^i) - d_p(U^i)| + |d_p(U^i) - d_q(U^i)| + |d_q(U^i) - c^{i+1}_q(U^i)|$.

We bound each of these three terms separately.

First, consider $|c^{i+1}_p(U^i) - d_p(U^i)|$. Now, $U^i + ADJ^i_p = D_p(u^i_p) = C^{i+1}_p(u^i_p)$. So

$|c^{i+1}_p(U^i) - d_p(U^i)| \leq |(c^{i+1}_p(U^i) - d_p(U^i)) - (c^{i+1}_p(U^i + ADJ^i_p) - d_p(U^i + ADJ^i_p))|$

$\leq \rho|ADJ^i_p|$, by Lemma 4-2

$\leq \rho((1 + \rho)(\beta + \varepsilon) + \rho\delta)$, by Lemma 4-6.

The same bound holds for the third term.

Finally, consider the middle term, $|d_p(U^i) - d_q(U^i)|$. We know that $d_p(U^i) = d_p(U^i + ADJ^i_p) - ADJ^i_p = u^i_p - ADJ^i_p$, and similarly for q.

$|d_p(U^i) - d_q(U^i)| = |(u^i_p - u^i_q) - (ADJ^i_p - ADJ^i_q)|$

$\leq \beta/2 + 2\varepsilon + 2\rho(2 + \rho)(\beta + \delta + \varepsilon)$, by Lemma 4-10.

Combining these three bounds, we get the required bound. ▮

Finally, we can show the second of our inductive properties, bounding the distance between times when clocks reach T^{i+1}.

Lemma 4-12: Let p, q be nonfaulty. Then $|t^{i+1}_p - t^{i+1}_q| \leq \beta$.

Proof: $|t^{i+1}_p - t^{i+1}_q|$

$= |c^{i+1}_p(T^{i+1}) - c^{i+1}_q(T^{i+1})|$

$\leq |(c^{i+1}_p(T^{i+1}) - c^{i+1}_q(T^{i+1})) - (c^{i+1}_p(U^i) - c^{i+1}_q(U^i))| + |c^{i+1}_p(U^i) - c^{i+1}_q(U^i)|$

$\leq 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon)) + \beta/2 + 2\varepsilon + 2\rho(3\beta + 2\delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon)$, by Lemmas 4-2 and 4-11.

The assumed upper bound on P implies that this expression is at most β. ∎

4.5.6 Bound on Message Arrival Time

In this subsection, we show that the third and final inductive assumption holds. That is, we show that messages arrive after the appropriate clocks have been set.

Lemma 4-13: Let p and q be nonfaulty. Then $t^{i+1}_q + \delta - \varepsilon > u^i_p$.

Proof: Since $t^{i+1}_q + \delta - \varepsilon \geq t^{i+1}_p - \beta + \delta - \varepsilon$, it suffices to show that

$t^{i+1}_p - u^i_p > \beta - \delta + \varepsilon$.

Now, $t^{i+1}_p - u^i_p \geq (P - (1 + \rho)(\beta + \delta + \varepsilon) - ADJ^i_p)/(1 + \rho)$ since the numerator represents the smallest possible difference in the values of the clock C^{i+1}_p at the two given real times.

But the lower bound on P implies that $P > 3(1 + \rho)(\beta + \varepsilon) + \rho\delta$. Also, the bound on the adjustment shows that $ADJ^i_p \leq (1 + \rho)(\beta + \varepsilon) + \rho\delta$. Therefore,

$t^{i+1}_p - u^i_p > (3(1 + \rho)(\beta + \varepsilon) + \rho\delta - (1 + \rho)(\beta + \delta + \varepsilon) - (1 + \rho)(\beta + \varepsilon) - \rho\delta) / (1 + \rho)$

$= \beta - \delta + \varepsilon$, as needed. ∎

Thus, we have shown that the three inductive hypotheses hold. Therefore, the claims made in this section for a particular i, in fact hold for all i.

4.6 Some General Properties

In this section, we state several consequences of the results proved in the preceding section.

First, we state a bound on the closeness with which the various clocks reach corresponding values.

Lemma 4-14: Let p, q be nonfaulty, $i \geq 0$. Assume that T is chosen so that $U^{i-1} \leq T \leq U^i$, if $i \geq 1$, or so that $T^0 \leq T \leq U^0$, if $i = 0$.

Then $|c^i_p(T) - c^i_q(T)| \leq \beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon)$.

Proof: *Basis:* $i = 0$. Then $T^0 \leq T \leq U^0$.

$|c^0_p(T) - c^0_q(T)| \leq |(c^0_p(T) - c^0_q(T)) - (c^0_p(T^0) - c^0_q(T^0))| + |c^0_p(T^0) - c^0_q(T^0)|$

$\leq 2\rho(T - T^0) + \beta$, by Lemma 4-2 and assumption 3

$\leq \beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon)$.

Induction: $i \geq 0$. Choose T with $U^{i-1} \leq T \leq U^i$.

$$|c^i_p(T) - c^i_q(T)| \leq |(c^i_p(T) - c^i_q(T)) - (c^i_p(U^{i-1}) - c^i_q(U^{i-1}))| + |c^i_p(U^{i-1}) - c^i_q(U^{i-1})|$$

$\leq 2\rho P + \beta/2 + 2\varepsilon + 2\rho(3\beta + 2\delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon)$, by Lemmas 4-2 and 4-11.

The upper bound on P implies the result. ▮

Next, we prove a bound for a nonfaulty process' (i + 1)-st clock, in terms of nonfaulty processes' i-th clocks.

Lemma 4-15: Let p be nonfaulty, $i \geq 0$. Then there exist nonfaulty processes, q and r, such that for $u^i_p \leq t \leq umax^i$,

$$C^i_q(t) - \alpha \leq C^{i+1}_p(t) \leq C^i_r(t) + \alpha,$$

where $\alpha = \varepsilon + \rho(4\beta + \delta + 5\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) + 2\rho^3(\beta + \delta + \varepsilon)$.

Proof: $C^{i+1}_p(t) = C^i_p(t) + T^i + \delta - AV^i_p$. Therefore, by Lemma 4-5 there are nonfaulty processes, q and r, for which

$$C^i_p(t) + T^i + \delta - ARR^i_p(q) \leq C^{i+1}_p(t) \leq C^i_p(t) + T^i + \delta - ARR^i_p(r).$$

We show the right-hand inequality first. Let $a = c^i_p(ARR^i_p(r))$, the real time at which the message arrives at p from r. Thus, $C^i_p(a) = ARR^i_p(r)$. Note that $C^i_r(a) \geq T^i + (1 - \rho)(\delta - \varepsilon)$.

$C^{i+1}_p(t) \leq C^i_p + T^i + \delta - ARR^i_p(r)$, from above

$\leq C^i_r(t) + C^i_p(a) - C^i_r(a) + T^i + \delta - ARR^i_p(r) + (C^i_p(t) - C^i_r(t)) - (C^i_p(a) - C^i_r(a))$

$\leq C^i_r(t) + C^i_p(a) - C^i_r(a) + T^i + \delta - ARR^i_p(r) + 2\rho(t - a)$, by Lemma 4-2 since $t > a$

$\leq C^i_r(t) + ARR^i_p(r) - T^i - (1 - \rho)(\delta - \varepsilon) + T^i + \delta - ARR^i_p(r) + 2\rho(t - a)$

$= C^i_r(t) + \varepsilon + \rho\delta - \rho\varepsilon + 2\rho(t - a)$.

It remains to bound $t - a$. The worst case occurs when $t = umax^i$. The longest possible elapsed real time between a particular nonfaulty process reaching T^i and U^i on the same clock is $(1 + \rho)^2(\beta + \delta + \varepsilon)$. Thus, $umax^i - tmin^i \leq \beta + (1 + \rho)^2(\beta + \delta + \varepsilon)$. But $a \geq tmin^i + \delta - \varepsilon$. Therefore, $t - a \leq \beta + (1 + \rho)^2(\beta + \delta + \varepsilon) - \delta + \varepsilon$

Thus, $C^{i+1}_p(t) \leq C^i_r(t) + \varepsilon + \rho\delta - \rho\varepsilon + 2\rho(\beta + (1 + \rho)^2(\beta + \delta + \varepsilon) - \delta + \varepsilon)$

$= C^i_r(t) + \varepsilon + \rho(4\beta + \delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) + 2\rho^3(\beta + \delta + \varepsilon)$

$< C^i_r(t) + \alpha$.

For the left-hand inequality, we see that $C^i_q(t) - \varepsilon - \rho\delta - \rho\varepsilon - 2\rho(t - a) \leq C^{i+1}_p(t)$, where $a = c^i_p(ARR^i_p(q))$. The factor $t - a$ is bounded exactly as before, so that we obtain:

$C^i_q(t) - \alpha \leq C^{i+1}_p(t)$. ∎

4.7 Agreement and Validity Conditions

We are now ready to show that the agreement and validity properties hold. The main effort is in restating bounds proved earlier concerning the closeness in real times when clocks reach the same value, in terms of the closeness of clock values at the same real time.

4.7.1 Agreement

The first lemma implies that the local times of two nonfaulty processes are close in those intervals where both use a clock with the same index.

Lemma 4-16: Let p, q be nonfaulty. Then

$$|C^i_p(t) - C^i_q(t)| \leq (1 + \rho)(\beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon))$$

for $\max\{u^{i-1}_p, u^{i-1}_q\} \leq t \leq \max\{u^i_p, u^i_q\}$, if $i \geq 1$,

and for $\min\{t^0_p, t^0_q\} \leq t \leq \max\{u^0_p, u^0_q\}$, if $i = 0$.

Proof: *Basis:* $i = 0$. Lemma 4-14 implies that

$$|c^i_p(T) - c^i_q(T)| \leq \beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon)$$

for all T, $U^{i-1} \leq T \leq U^i$ if $i \geq 1$ and for all T, $T^0 \leq T \leq U^0$ if $i = 0$. Then Lemma 4-3 immediately implies the needed result for $i = 0$.

Induction: $i \geq 1$. Lemma 4-3 implies the result for all t with

$$\min\{c^i_p(U^{i-1}), c^i_q(U^{i-1})\} \leq t \leq \max\{u^i_p, u^i_q\}.$$

It remains to show the bound for t with

$$\max\{u^{i-1}_p, u^{i-1}_q\} \leq t < \min\{c^i_p(U^{i-1}), c^i_q(U^{i-1})\}.$$

Without loss of generality, assume that $c^i_p(U^{i-1}) \leq c^i_q(U^{i-1})$, so that the minimum is equal to $c^i_p(U^{i-1})$.

$$|C^i_p(t) - C^i_q(t)| \leq |(C^i_p(t) - C^i_q(t)) - (C^i_p(c^i_p(U^{i-1})) - C^i_q(c^i_p(U^{i-1})))|$$
$$+ |C^i_p(c^i_p(U^{i-1})) - C^i_q(c^i_p(U^{i-1}))|$$

The first term, by Lemma 4-2, is at most $2\rho(c^i_p(U^{i-1}) - t)$. Since $t \geq \max\{u^{i-1}_p, u^{i-1}_q\} \geq u^{i-1}_p \geq c^{i-1}_p(U^{i-1})$, we have

$$2\rho(c^i_p(U^{i-1}) - t) \leq 2\rho(c^i_p(U^{i-1}) - c^{i-1}_p(U^{i-1})).$$

Since $c^{i-1}_p(U^{i-1}) = c^i_p(T)$ for some T with $|T - U^{i-1}| \leq |ADJ^i_p|$, this quantity is

$\leq 2\rho|c^i_p(U^{i-1}) - c^i_p(T)|$

$\leq 2\rho(1 + \rho)|U^{i-1} - T|$, by Lemma 4-1

$\leq 2\rho(1 + \rho)|ADJ^i_p|$

$\leq 2\rho(1 + \rho)((1 + \rho)(\beta + \varepsilon) + \rho\delta)$, by Lemma 4-6.

To bound the second term we note that Lemma 4-11 implies that

$|c^i_p(U^{i-1}) - c^i_q(U^{i-1})| \leq \beta/2 + 2\varepsilon + 2\rho(3\beta + 2\delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) = \alpha,$

and so Lemma 4-3, with $T_1 = T_2 = U^{i-1}$, implies that

$|C^i_p(c^i_p(U^{i-1})) - C^i_q(c^i_p(U^{i-1}))| \leq (1 + \rho)\alpha.$

The assumed lower bound on β gives the result that

$2\rho(1 + \rho)((1 + \rho)(\beta + \varepsilon) + \rho\delta) + (1 + \rho)\alpha \leq (1 + \rho)(\beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon))$ ∎

Here is the main result, bounding the error in the synchronization at any time.

Theorem 4-17: The algorithm guarantees γ-agreement,

where $\gamma = \beta + \varepsilon + \rho(7\beta + 3\delta + 7\varepsilon) + 8\rho^2(\beta + \delta + \varepsilon) + 4\rho^3(\beta + \delta + \varepsilon)$.

Proof: The result for intervals in which the processes use clocks with the same indices has been covered in the preceding lemma. The expression in the statement of that lemma simplifies to

$\beta + \rho(3\beta + 2\delta + 2\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) + 2\rho^3(\beta + \delta + \varepsilon),$

which is less than γ.

Next, we must consider the case where one of the processes has changed to a new clock, while the other still retains the old clock. Consider $|C^{i+1}_p(t) - C^i_q(t)|$ for some t with $u^i_p \leq t \leq u^i_q$. Lemma 4-15 implies that there exist nonfaulty processes r and s such that

$C^i_r(t) - \alpha \leq C^{i+1}_p(t) \leq C^i_s(t) + \alpha,$

where $\alpha = \varepsilon + \rho(4\beta + \delta + 5\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) + 2\rho^3(\beta + \delta + \varepsilon)$.

$|C^{i+1}_p(t) - C^i_q(t)| \leq \alpha + \max\{|C^i_r(t) - C^i_q(t)|, |C^i_s(t) - C^i_q(t)|\}$

$\leq \alpha + (1 + \rho)(\beta + 2\rho(1 + \rho)(\beta + \delta + \varepsilon))$, by the preceding lemma

$= \beta + \varepsilon + \rho(7\beta + 3\delta + 7\varepsilon) + 8\rho^2(\beta + \delta + \varepsilon) + 4\rho^3(\beta + \delta + \varepsilon)$, as needed. ∎

In some applications, it may never be the case that clocks with different indices are compared, perhaps because use of the clocks for processing ceases during the interval in which confusion is possible. In that case, the closeness of synchronization achieved by Algorithm 4-1 is given by Lemma 4-16, and is approximately $\beta + \rho(3\beta + 2\delta + 2\varepsilon)$. This value is more than ε less than the bound obtained when clocks with different indices must be compared.

Now we can sketch why it is reasonable for β to be approximately $4\varepsilon + 4\rho P$, as mentioned at the end of Section 4.5.1. Assume P is fixed. The i-th clocks reach T^i within β of each other. After the processes reset their clocks, the new clocks reach U^i within $\beta/2 + 2\varepsilon$ (ignoring ρ terms). By the end of the round, the clocks reach T^{i+1} within about $\beta/2 + 2\varepsilon + 2\rho P$ of each other, because of drift. This quantity must be at most β. The inequality $\beta/2 + 2\varepsilon + 2\rho P \leq \beta$ yields $\beta \geq 4\varepsilon + 4\rho P$.

Suppose we alter the algorithm so that during each round, the processes exchange clock values k times instead of just once. Then we get $\beta/2^k + (4 - 2^{2-k})\varepsilon + 2\rho P \leq \beta$, which simplifies to $\beta \geq 4\varepsilon + 2\rho P(2^k/(2^k-1))$. It appears that $\beta \geq 4\varepsilon + 2\rho P$ is approachable.

If the number of processes, n, increases while f, the number of faulty processes remained fixed, a greater closeness of synchronization can be achieved by modifying Algorithm 4-1 so that it computes the mean instead of the midpoint of the range of values.

As in [1], we show that the convergence rate of algorithms that use the mean instead of the midpoint is roughly f/(n–2f).

The result is based on the following lemma concerning multisets.

Lemma 4-18: Let U, V, and W be multisets such that $|U| = |V| = n \geq 3f + 1$ and $|W| = n - f$. If $d_x(W,U) = d_x(W,V) = 0$, then

$$|\mathrm{mean}(\mathrm{reduce}(U)) - \mathrm{mean}(\mathrm{reduce}(V))| \leq \mathrm{diam}(W)f/(n-2f) + 2x.$$

The analysis of the modified Algorithm 4-1 parallels that just presented. However, the upper bound on P becomes

$$P \leq \beta(n-3f)/(n-2f)2\rho - \varepsilon/\rho - \rho(\beta + \delta + \varepsilon) - 2\beta - \delta - 2\varepsilon.$$

This bound implies $\beta \geq 2(n-2f)(\varepsilon + \rho P)/(n-3f)$, which approaches $\beta \geq 2\varepsilon + 2\rho P$ as n approaches infinity.

We now demonstrate that this bound is reasonable. After updating the clock and then waiting until the clocks reach the next T^i, the clocks must still be within β, giving $f\beta/(n-2f) + 2\varepsilon + 2\rho P \leq$

β, which implies $\beta \geq (2\varepsilon + 2\rho P)(n-2f)/(n-3f)$, which approaches $2\varepsilon + 2\rho P$ as n approaches infinity.

4.7.2 Validity

Next, we show the validity condition. The first lemma bounds the values of the zero-index clocks.

Lemma 4-19: $T^0 + (1-\rho)(t-t^0_p) \leq C^0_p(t) \leq T^0 + (1+\rho)(t-t^0_p)$ for $t \geq t^0_p$.

Proof: By Lemma 4-1. ∎

The next lemma is the main one.

Lemma 4-20: Let p be nonfaulty, $i \geq 0$. Then

$$(1-\rho)(t-tmax^0) + T^0 - i\varepsilon \leq C^i_p(t) \leq (1+\rho)(t-tmin^0) + T^0 + i\varepsilon$$

for all $t \geq u^{i-1}_p$ if $i \geq 1$, and for all $t \geq t^0_p$ if $i = 0$.

Proof: We proceed by induction on i. When proving the result for i + 1, we will assume the result for i, for all executions of the algorithm (rather than just the execution in question).

Basis: $i = 0$. This case follows immediately by Lemma 4-19.

Induction: Assume the result has been shown for i and show it for i + 1.

We argue the right-hand inequality first. The left-hand inequality is entirely analogous.

Assume in contradiction that we have a particular execution in which $C^{i+1}_p(t) > (1+\rho)(t - tmin^0) + T^0 + (i+1)\varepsilon$ for some $t \geq u^i_p$. Then by the limitations on rates of clocks, it is clear that $C^{i+1}_p(u^i_p) > (1+\rho)(u^i_p - tmin^0) + T^0 + (i+1)\varepsilon$.

Recall that p resets its clock at real time u^i_p, by adding $T^i + \delta - AV^i_p$. In this case, the inductive hypothesis implies that the adjustment must be an increment.

By Lemma 4-5, this increment is $\leq T^i + \delta - ARR^i_p(q)$ for some nonfaulty q. Therefore,

$$C^i_p(u^i_p) + T^i + \delta - ARR^i_p(q) > (1+\rho)(u^i_p - tmin^0) + T^0 + (i+1)\varepsilon.$$

Next, we claim that if p had done the adjustment just when the message arrived from q rather than waiting till real time u^i_p, the bound would still have been exceeded. That is, $ARR^i_p(q) + T^i + \delta - ARR^i_p(q) > (1+\rho)(t' - tmin^0) + T^0 + (i+1)\varepsilon$, where $t' = c^i_p(ARR^i_p(q))$. (This again follows by the limits on the rates of clocks.) Thus,

$$T^i + \delta > (1+\rho)(t' - tmin^0) + T^0 + (i+1)\varepsilon.$$

Now consider an alternative execution of the algorithm in which everything is exactly like the one we have been describing, except that immediately after q sends out clock reading T^i, q's clock C^i_q begins to move at rate 1. This change cannot affect p's $(i+1)$-st clock because q doesn't send any more messages until t^{i+1}_q, and these messages aren't received until after the time when p sets its $(i+1)$-st clock.

By the lower bound on message delays, q's message to p took at least $\delta - \varepsilon$ time. Then at real time t' (defined above), we have $C^i_q(t') \geq T^i + \delta - \varepsilon$. But then $C^i_q(t') > (1+\rho)(t' - tmin^0) + T^0 + i\varepsilon$.

But then the inductive hypothesis is violated, since t', the time when p receives q's T^i message, is greater than or equal to u^{i-1}_q, the time when q sets its round i clock. ∎

Now, we can state the validity condition. Let $\varphi = (P - (1+\rho)(\beta + \varepsilon) - \rho\delta) / (1+\rho)$. This is the size of the shortest round in real time since the amount of clock time elapsed during a round is at least P minus the maximum adjustment.

Theorem 4-21: The algorithm preserves $(\alpha_1, \alpha_2, \alpha_3)$-validity,

where $\alpha_1 = 1 - \rho - \varepsilon/\varphi$, $\alpha_2 = 1 + \rho + \varepsilon/\varphi$, and $\alpha_3 = \varepsilon$.

Proof: We must show for all $t \geq t^0_p$ and all nonfaulty p that

$$\alpha_1(t - tmax^0) + T^0 - \alpha_3 \leq L_p(t) \leq \alpha_2(t - tmin^0) + T^0 + \alpha_3.$$

We know from the preceding lemma that for $i \geq 0$, $t \geq u^{i-1}_p$ (or t^0_p), and nonfaulty p

$$(1-\rho)(t - tmax^0) + T^0 - i\varepsilon \leq C^i_p(t) \leq (1+\rho)(t - tmin^0) + T^0 + i\varepsilon.$$

Since $L_p(t)$ is equal to $C^i_p(t)$ for some i, we just need to convert i into an expression in terms of t, etc. An upper bound on i is $1 + (t - tmax^0)/\varphi$. Then

$$(1+\rho)(t - tmin^0) + T^0 + i\varepsilon \leq (1+\rho)(t - tmin^0) + T^0 + (1 + (t - tmax^0)/\varphi)\varepsilon$$

$$\leq (1 + \rho + \varepsilon/\varphi)(t - tmin^0) + T^0 + \varepsilon, \text{ since } tmin^0 \leq tmax^0,$$

and that

$$(1-\rho)(t - tmax^0) + T^0 - i\varepsilon \geq (1-\rho)(t - tmax^0) + T^0 - (1 + (t - tmax^0)/\varphi)\varepsilon$$

$$\geq (1 - \rho - \varepsilon/\varphi)(t - tmax^0) + T^0 - \varepsilon.$$

The result follows. ∎

4.8 Reintegrating a Repaired Process

Our algorithm can be modified to allow a faulty process which has been repaired to synchronize its clock with the other nonfaulty processes. Let p be the process to be reintegrated into the system. During some round i, p will gather messages from the other processes and perform the same averaging procedure described previously to obtain a value for its correction variable such that its clock becomes synchronized. Since p's clock is now synchronized, it will reach T^{i+1} within β of every other nonfaulty process. At that point, p is no longer faulty and rejoins the main algorithm, sending out T^{i+1} messages.

We assume that p can awaken at an arbitrary time during an execution, perhaps during the middle of a round. It is necessary that p identify an appropriate round i at which it can obtain all the T^i messages from nonfaulty processes. Since p might awaken during the middle of a round, p will orient itself by observing the arriving messages. More specifically, p seeks an i such that f T^{i-1} messages arrive within an interval of length at most $(1 + \rho)(\beta + 2\varepsilon)$ as measured on its clock. There will always be such an i because all messages from nonfaulty processes for each round arrive within $\beta + 2\varepsilon$ real time of each other, and thus within $(1 + \rho)(\beta + 2\varepsilon)$ clock time. At the same time as p is orienting itself, it is collecting T^j messages, for all j.

Assuming that p itself is still counted as one of the faulty processes, at least one of the f arriving messages must be from a nonfaulty process. Thus, p knows that round $i - 1$ is in progress or has just ended, and that it should use T^i messages to update its clock.

Now p collects only T^i messages. It must wait $(1 + \rho)(\beta + 2\varepsilon + (1 + \rho)(P + (1 + \rho)(\beta + \varepsilon) + \rho\delta)$, as measured on its clock, after receiving the f-th T^{i-1} message in order to guarantee that it has received T^i messages from all nonfaulty processes. The maximum amount of real time p must wait, $(\beta + 2\varepsilon + (1 + \rho)(P + (1 + \rho)(\beta + 2\varepsilon) + \rho\delta)$, elapses if the f-th T^{i-1} message is from a nonfaulty process q and it took $\delta - \varepsilon$ time to arrive, if q's round $i - 1$ lasts a long as possible, $(1 + \rho)(P + (1 + \rho)(\beta + \varepsilon) + \rho\delta)$ (because its clock is slow and it adds the maximum amount to its clock), and if there is a nonfaulty process r that is β behind q in reaching T^i and its T^i message to p takes $\delta + \varepsilon$. The process waits this maximum amount of time multiplied by $(1 + \rho)$ to account for a fast clock.

(Some extra bookkeeping in the algorithm is necessitated by the fact that T^i messages from nonfaulty processes can arrive at p before p has received the f-th T^{i-1} message. This scenario shows why: Suppose p receives the first T^{i-1} message at real time a, it is from a nonfaulty process q, and its delay is $\delta + \varepsilon$, and that the f-th T^{i-1} message is received $\beta + 2\varepsilon$ after the first one. Also suppose that q's round $i - 1$ is as short as possible in real time, $P - (1 + \rho)(\beta + \varepsilon) - \rho\delta) / (1 + \rho)$, that there is a nonfaulty process r that begins round i β before q does, and that r's T^i message to p arrives at real time b and has delay $\delta - \varepsilon$.

We show that $b < a + \beta + 2\varepsilon$, implying that the T^i message is received before the f-th T^{i-1}

message.

$b = t^i_r + \delta - \varepsilon$

$= t^i_q - \beta + \delta - \varepsilon$

$= t^{i-1}_q + (P - (1 + \rho)(\beta + \varepsilon) - \rho\delta) / (1 + \rho) - \beta + \delta - \varepsilon$

$> t^{i-1}_q + ((1 + \rho)(3\beta + 3\varepsilon) + \rho\delta - (1 + \rho)(\beta + \varepsilon) \cdot \rho\delta) / (1 + \rho) - \beta + \delta - \varepsilon$, by lower bound on P

$= t^{i-1}_q + \beta + \delta + \varepsilon$

$= a - \delta - \varepsilon + \beta + \delta + \varepsilon.$

Thus, $b > a + \beta$. However, if P is very close to the lower bound, then b is approximately $a + \beta$, which is less than $a + \beta + 2\varepsilon$.)

Immediately after p determines it has waited long enough, it carries out the averaging procedure and determines a value for its correction variable.

We claim that p reaches T^{i+1} on its new clock within β of every other nonfaulty process. First, observe that it does not matter that p's clock begins initially unsynchronized with all the other clocks; the arbitrary clock will be compensated for in the subtraction of the average arrival time. Second, observe that it does not matter that p is not sending out a T^i message; p is being counted as one of the faulty processes, which could always fail to send a message. (Processes do not treat themselves specially in our algorithm, so it does not matter that p fails to receive a message from itself.) Finally, observe that it does not matter that p adjusts its correction variable whenever it is ready (rather than at the time specified for correct processes in the ordinary algorithm). The adjustment is only the addition of a constant, so the (additive) effect of the change is the same in either case.

We want to ensure that when a process that is reintegrating itself into the system finishes collecting T^i messages and updates its clock, this new clock hasn't already passed T^{i+1}. The reason for ensuring this is that the process is supposed to be nonfaulty by T^{i+1} and send out its clock value at that time.

The code is in Figure 4-2.

INFO is an array, each entry of which is a set of (process name, clock time) pairs. When a T^i

```
beginstep(u)
do forever
     if u = (T^i,q) and (q,T) ∉ INFO[i] for any T then
          INFO[i] := INFO[i] ∪ {(q,NOW)}
          if |{(q,T) ∈ INFO[i]:  q is any process and
             T ≥ NOW - (1 + ρ)(β + 2ε)}| = f
               then exit endif
          endif
     endstep
     beginstep(u)
     enddo

/* p knows it should use round i values */

do for each (q,T) ∈ INFO[i]
     ARR[q] := T
     enddo
set-timer(NOW + (1 + ρ)(β + 2ε + (1 + ρ)(P + (1 + ρ)(β + ε) + ρδ)))
endstep

beginstep(u)
while u = (T^i,q) for the chosen i do
     ARR[q] := NOW
     endstep
     beginstep(u)
     endwhile

/* fall out of loop when timer goes off */

AV := mid(reduce(ARR))
ADJ := T^i + δ - AV
CORR := CORR + ADJ
set-timer(T^i + P)
endstep

/* switch to Algorithm 4-1 */
```

Figure 4-2: Algorithm 4-2, Reintegrating a Repaired Process

message arrives from process q, p checks that q hasn't already sent it a T^i message. If not, then q's name and the receiving time are added to the set of senders of T^i, INFO[i]. If f distinct T^i messages have been received within the last $(1 + \rho)(\beta + 2\varepsilon)$ time, then p knows that it should use T^i messages to update its clock.

The current lower bound on P, the round length, is not large enough to ensure that when the reintegrating process finishes collecting T^i messages and updates its clock, this new clock hasn't already passed T^{i+1}.

There are two ways to solve this problem:

1. make the minimum P approximately three times as large as it currently must be;

2. have the process send out its clock value at T^{i+2}. It can be collecting T^{i+1} messages all along, but now it knows a tighter bound on when to stop collecting them (since its $(i+1)$-st clock is synchronized with the other nonfaulty processes' clocks). This will work as long as the time at which it stops collecting T^i messages isn't after the process' $(i+2)$-nd clock has reached T^{i+2}.

Now we show that P must be about three times as large as the previous lower bound in order to prevent the reintegrating process from waiting too long before updating its clock. The actual criterion we use is that the process must update its clock at least β before any other nonfaulty process' $(i+1)$-st clock reaches T^{i+1}. (Since the process' new clock is synchronized with those of the nonfaulty processes, it will not reach T^{i+1} more than β before any other nonfaulty clock does.)

Let p be a process being reintegrated during round i and let t be the real time when p stops collecting T^i messages

Lemma 4-22: If $t \leq c^{i+1}_q(T^{i+1}) - \beta$ for any nonfaulty process q, then

$$P > (6\beta + \delta + 9\varepsilon + \rho(8\beta + 3\delta + 16\varepsilon) + \rho^2(6\beta + \delta + 14\varepsilon) + \rho^3(4\beta + 3\delta + 8\varepsilon) + \rho^4(\beta + \delta + 2\varepsilon)) / (1 - 5\rho - 3\rho^2 - \rho^3).$$

Proof: The worst case occurs if p waits as long as possible to finish collecting T^i messages and another nonfaulty process q reaches T^{i+1} as soon as possible.

Suppose p receives the first T^{i-1} message at real time t', and the f-th T^{i-1} message at $t' + (1 + \rho)^2(\beta + 2\varepsilon)$ (because its clock is slow). According to the reintegration algorithm, p will then wait $(1 + \rho)(\beta + 2\varepsilon + (1 + \rho)(P + (1 + \rho)(\beta + 2\varepsilon) + \rho\delta))$ on its clock, which means it will wait $(1 + \rho)$ times as long in real time.

Thus, $t = t' + (1 + \rho)^2(2\beta + 4\varepsilon + (1 + \rho)(P + (1 + \rho)(\beta + 2\varepsilon) + \rho\delta))$.

Now assume that the first T^{i-1} message received by p was from a nonfaulty process q and that it took $\delta + \varepsilon$ time to arrive. Thus $c^{i-1}_q(T^{i-1}) = t' - \delta - \varepsilon$. If round $i - 1$ and round i both take the shortest amount of real time, $(1 - \rho)(P - (1 + \rho)(\beta + \varepsilon) - \rho\delta)$, then

$$c^{i+1}_q(T^{i+1}) = c^{i-1}_q(T^{i-1}) + 2(1 - \rho)(P - (1 + \rho)(\beta + \varepsilon) - \rho\delta).$$

We want to ensure that $c^{i+1}_q(T^{i+1}) - t \geq \beta$, i.e.,

$$t' - \delta - \varepsilon + 2(1 - \rho)(P - (1 + \rho)(\beta + \varepsilon) - \rho\delta)$$
$$- t' - (1 + \rho)^2(2\beta + 4\varepsilon + (1 + \rho)(P + (1 + \rho)(\beta + 2\varepsilon) + \rho\delta)) \geq \beta.$$

This inequality simplifies to the stated bound. ∎

This new lower bound on P is about three times the size of the previous one, which was

$P > 2\beta + \delta + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon)$.

If increasing the lower bound on P is unacceptable, the second solution can be employed. Its drawback is that now it will take longer for a process to be reintegrated. A similar argument to the above shows that in order to guarantee that p finishes collecting T^i messages at least β before any nonfaulty process reaches T^{i+2}, we must have

$P \geq (5\beta + \delta + 10\varepsilon + 2\rho(5\beta + 2\delta + 9\varepsilon)) / (2 - 4\rho)$, ignoring ρ^2 terms.

This lower bound is fairly close to the original one. For absolute certainty that the original lower bound will suffice, the process can wait until T^{i+3}.

Chapter Five

Establishing Synchronization

5.1 Introduction

In this chapter we present an algorithm to synchronize clocks in a distributed system of processes, assuming the clocks initially have arbitrary values. The algorithm handles arbitrary failures of the processes and clock drift. We envision the processes running this algorithm until the desired degree of synchronization is obtained, and then switching to the maintenance algorithm described in the previous chapter.

5.2 The Algorithm

5.2.1 General Description

The structure of the start-up algorithm is similar to that of the algorithm which maintains synchronization. It runs in rounds. During each round, the processes exchange clock values and use the same fault-tolerant averaging function as before to calculate the corrections to their clocks. However, each round contains an additional phase, in which the processes exchange messages to decide that they are ready to begin the next round. This method of beginning rounds stands in contrast to that used by the maintenance algorithm, in which rounds begin when local clocks reach particular values. A more detailed description follows.

Nonfaulty processes will begin each round within real time $\delta + 3\epsilon$ of each other. Each nonfaulty process begins the algorithm, and its round 0, as soon as it first receives a message. (It will be shown that this must be within $\delta + 3\epsilon$.) At the beginning of each round, each nonfaulty process p broadcasts its local time. Then p waits a certain length of time guaranteed to be long enough for it to receive a similar message from each nonfaulty process. At the end of this waiting interval, p calculates the adjustment it will make to its clock at the current round, but does not make the adjustment yet.

Then p waits a second interval of time before sending out additional messages, to make sure that these new messages are not received before the other nonfaulty processes have reached the end of their first waiting intervals. At the end of its second waiting interval, p broadcasts a READY message indicating that it is ready to begin the next round. However, if p receives $f + 1$ READY messages during its second waiting interval, it terminates its second interval early, and goes ahead and broadcasts READY. As soon as p receives $n - f$ READY messages, it updates the clock according to the adjustment calculated earlier, and begins its next round by broadcasting its new clock value. (This algorithm uses some ideas from [3].)

A process need only keep clock differences for one round at a time. The waiting intervals are designed so that during round i a nonfaulty process p will not receive a READY message from another nonfaulty process until p has finished collecting round i clock values. Round $i + 1$ clock values are not broadcast until after READY is broadcast, so p will certainly not receive round $i + 1$ clock values until after it has finished collecting round i clock values. However, round $i + 1$ clock values might arrive during the second waiting interval and while the process is collecting READY messages. As a result, the adjustment is calculated at the end of the first waiting interval and the difference for any round $i + 1$ clock value received during round i is decremented by the amount of the adjustment.

5.2.2 Code for an Arbitrary Process

Global constants: δ, ε, ρ, n, f: as usual.

Local variables (all initially arbitrary):

- T: clock time at which current round began.
- U: clock time at which the first waiting period is to end.
- V: clock time at which the second waiting period is to end.
- DIFF: array of clock differences between other processes and this one for current round.
- SENT-READY: set of processes from whom READY messages have been received in current round.
- CORR: correction variable.
- A: adjustment to clock.

The code is in Figure 5-1.

```
beginstep(w)
do forever  /* each iteration is a round */
     T := NOW
     broadcast(T)
     U := T + (1 + ρ)(2δ + 4ε)
     set-timer(U)

/* first waiting interval:  collect clock values */

     while ~(w = TIMER & NOW = U) do
          if w = (m,q) then  DIFF[q] := m + δ - NOW endif
          endstep
          beginstep(w)
          endwhile

/* end of first waiting interval */

     A := mid(reduce(DIFF))
     V := U + (1 + ρ)(4ε + 4ρ(δ + 2ε) + 2ρ²(δ + 2ε))
     set-timer(V)
     SENT-READY := Ø

/* second waiting interval:  collect READY messages and clock values
   for next round */

     while ~(w = TIMER & NOW = V) do
          if w = (READY,q) then
               SENT-READY := SENT-READY ∪ {q}
               if |SENT-READY| = f + 1 then exit endif
          elseif w = (m,q) then DIFF[q] := m + δ - NOW endif
          endstep
          beginstep(w)
          endwhile

/* end of second waiting interval due to timer or f + 1 READY messages */

     broadcast(READY)
     endstep
     beginstep(w)

/* collect n - f READY messages and next round clock values */

     while true do
          if w = (READY,q) then
               SENT-READY := SENT-READY ∪ {q}
               if |SENT-READY| = n - f then exit endif
          elseif w = (m,q) then DIFF[q] := m + δ - NOW endif
          endstep
          beginstep(w)
          endwhile

/* update clock and begin next round */

     DIFF := DIFF - A
     CORR := CORR + A
     endstep
     beginstep(w)
     enddo
```

Figure 5-1: Algorithm 5-1, Establishing Synchronization

5.3 Analysis

We will use the following notation in addition to that introduced already.

- $VAL^i_p(q)$ is the value of q's round i message to p.
- $DIFF^i_p(q) = VAL^i_p(q) + \delta - ARR^i_p(q)$, p's estimate of the difference between p's and q's clocks.
- $DIFF^i_p$ is the multiset of $DIFF^i_p(q)$ values.
- t^i_p is the real time when p begins round i.
- u^i_p is the real time when p begins the second waiting interval during round i.
- v^i_p is the real time when p sends READY during round i (and thus ends the second waiting interval).
- $arr^i_p(q)$ is the real time when p first receives a round i clock value from q.
- $rdy^i_p(q)$ is the real time when p first receives READY from q during round i.
- $tmax^i = max\{t^i_p\}$ for p nonfaulty, the latest real time when a nonfaulty process begins round i.
- $B^i = max\{|C^i_p(tmax^i) - C^i_q(tmax^i)|\}$ for p and q nonfaulty, the maximum difference between nonfaulty clock values at $tmax^i$.

Note that $tmax^i$ has a slightly different meaning from that in Chapter 4.

From now on, terms of order ρ^2 and higher will be ignored. Since 10^{-6} seconds is an often quoted reasonable value for ρ [5, 7, 9], terms of order ρ^2 are negligible. The second-order terms in the assignment to V in line 13 of the code are needed for strict correctness, but will not appear in the analysis.

Lemma 5-2 proves together inductively that the time between two nonfaulty processes beginning a round is bounded, and that when a nonfaulty process q receives READY from another nonfaulty process, q has already finished the first waiting period. First we show a preliminary lemma needed by Lemma 5-2.

Lemma 5-1: Let $i \geq 0$, p and q be any nonfaulty processes, and r be the first nonfaulty process to send READY at round i. Then

$rdy^i_q(p) \geq u^i_q - (t^i_q - t^i_r) + \delta + 3\varepsilon$.

Proof: Since r is the first nonfaulty process to send READY, it doesn't send until its full second waiting interval has elapsed. Then

$rdy^i_q(p) \geq v^i_p + \delta - \varepsilon$

$\geq v^i_r + \delta - \varepsilon$

$\geq t^i_r + (2\delta + 4\varepsilon) + (4\varepsilon + 4\rho(\delta + 2\varepsilon)) + \delta - \varepsilon$, by definition of v^i_r and the upper bound on the drift rate

$= t^i_r + 3\delta + 7\varepsilon + 4\rho\delta + 8\rho\varepsilon,$

and

$u^i_q \leq t^i_r + (t^i_q - t^i_r) + (u^i_q - t^i_q)$

$\leq t^i_r + (t^i_q - t^i_r) + (1 + \rho)^2(2\delta + 4\varepsilon)$, by definition of u^i_q and the lower bound on the drift rate

$= t^i_r + (t^i_q - t^i_r) + 2\delta + 4\varepsilon + 4\rho\delta + 8\rho\varepsilon.$

Thus, $t^i_r \geq u^i_q - (t^i_q - t^i_r) - 2\delta - 4\varepsilon - 4\rho\delta - 8\rho\varepsilon$, implying

$rdy^i_q(p) \geq u^i_q - (t^i_q - t^i_r) - 2\delta - 4\varepsilon - 4\rho\delta - 8\rho\varepsilon + 3\delta + 7\varepsilon + 4\rho\delta + 8\rho\varepsilon$

$= u^i_q - (t^i_q - t^i_r) + \delta + 3\varepsilon.$ ∎

Lemma 5-2: For any nonfaulty processes p and q and any $i \geq 0$,

(a) $|t^i_p - t^i_q| \leq \delta + 3\varepsilon$, and

(b) $rdy^i_q(p) \geq u^i_q$.

Proof: We proceed by induction on i.

Basis: $i = 0$.

(a) $|t^0_p - t^0_q| \leq \delta + \varepsilon$, because as soon as p wakes up, it sends its round 0 message to all other processes. The receipt of this message, which occurs at most $\delta + \varepsilon$ later, causes q to begin round 0, if it hasn't already done so.

(b) Let r be the first nonfaulty process to send READY at round 0. By Lemma 5-1,

$rdy^0_q(p) \geq u^0_q - (t^0_q - t^0_r) + \delta + 3\varepsilon$

$\geq u^0_q - (\delta + \varepsilon) + \delta + 3\varepsilon$, by part (a)

$> u^0_q$.

Induction: Assume for $i - 1$ and show for i.

(a) Let s be the first nonfaulty process to begin round i. Then s receives $n - f$ READY messages during its round $i - 1$ (after u^{i-1}_s). At least $n - 2f$ of them are from nonfaulty processes by part (b) of the induction hypothesis. These $n - 2f$ nonfaulty processes also send READY messages to all the other processes. By $t^i_s + 2\varepsilon$, every nonfaulty process receives at least $n - 2f \geq f + 1$ READY messages and broadcasts READY. Thus q receives $n - f$ READY messages by $t^i_s + 2\varepsilon + \delta + \varepsilon$. Thus,

$$t^i_q \leq t^i_s + \delta + 3\varepsilon$$

$$\leq t^i_p + \delta + 3\varepsilon, \text{ by choice of s,}$$

which implies $t^i_q - t^i_p \leq \delta + 3\varepsilon$.

By reversing the roles of p and q in the above argument, we obtain $t^i_p - t^i_q \leq \delta + 3\varepsilon$.

(b) Let r be the first nonfaulty process to send READY at round i. By Lemma 5-1,

$$rdy^i_q(p) \geq u^i_q - (t^i_q - t^i_r) + \delta + 3\varepsilon$$

$$\geq u^i_q - (\delta + 3\varepsilon) + \delta + 3\varepsilon, \text{ by part (a)}$$

$$= u^i_q. \blacksquare$$

Next we show that a process waits a sufficient length of time to receive clock values from all nonfaulty processes before beginning the second waiting interval in a round.

Lemma 5-3: Let p and q be nonfaulty, and $i \geq 0$. Then $arr^i_p(q) \leq u^i_p$.

Proof: By the lower bound on the drift rate, $u^i_p \geq t^i_p + 2\delta + 4\varepsilon$. Lemma 5-2 implies that q sends its round i clock value by $t^i_p + \delta + 3\varepsilon$. Thus $arr^i_p(q) \leq t^i_p + 2\delta + 4\varepsilon \leq u^i_p$. ∎

The next two lemmas bound how long a round can last for one process. First we bound how long a process must wait after sending READY to receive $n - f$ READY messages.

Lemma 5-4: For p nonfaulty and $i \geq 0$, $t^{i+1}_p - v^i_p \leq 2\delta + 4\varepsilon + 4\rho(\delta + 4\varepsilon)$.

Proof: The worst case occurs if p is as far ahead of the other nonfaulty processes as possible, its clock is fast, the other clocks are slow, and the slow processes' READY messages take as long as possible to arrive. However, as soon as they arrive, p begins the next round. Let q be one of the slow nonfaulty processes.

$$t^{i+1}_p - v^i_p = (t^{i+1}_p - v^i_q) + (v^i_q - u^i_q) + (u^i_q - t^i_q) + (t^i_q - t^i_p) - (v^i_p - u^i_p) - (u^i_p - t^i_p)$$

$$\leq (\delta + \varepsilon) + (1 + \rho)^2(4\varepsilon + 4\rho(\delta + 2\varepsilon)) + (1 + \rho)^2(2\delta + 4\varepsilon) + (\delta + 3\varepsilon)$$
$$- (4\varepsilon + 4\rho(\delta + 2\varepsilon)) - (2\delta + 4\varepsilon)$$

$$= 2\delta + 4\varepsilon + 4\rho(\delta + 4\varepsilon), \text{ ignoring } \rho^2 \text{ terms. } \blacksquare$$

Lemma 5-5: For any nonfaulty process p and any $i \geq 0$,

$$t^{i+1}_p - t^i_p \leq 4\delta + 12\varepsilon + 4\rho(3\delta + 10\varepsilon).$$

Proof: $t^{i+1}_p - t^i_p = (t^{i+1}_p - v^i_p) + (v^i_p - u^i_p) + (u^i_p - t^i_p)$

$\leq 2\delta + 4\varepsilon + 4\rho(\delta + 4\varepsilon) + (v^i_p - u^i_p) + (u^i_p - t^i_p)$, by Lemma 5-4

$\leq 2\delta + 4\varepsilon + 4\rho(\delta + 4\varepsilon) + (1 + \rho)^2(4\varepsilon + 4\rho(\delta + 2\varepsilon)) + (1 + \rho)^2(2\delta + 4\varepsilon)$

$= 4\delta + 12\varepsilon + 4\rho(3\delta + 10\varepsilon)$. ∎

Now we give an upper bound on how far apart $tmax^i$ and $tmax^{i+1}$ can be.

Lemma 5-6: For any $i \geq 0$,

$tmax^{i+1} - tmax^i \leq 4\delta + 12\varepsilon + 4\rho(3\delta + 10\varepsilon)$.

Proof: Let p be the nonfaulty process such that $t^{i+1}_p = tmax^{i+1}$. Then

$tmax^{i+1} - tmax^i = t^{i+1}_p - tmax^i \leq t^{i+1}_p - t^i_p$

$\leq 4\delta + 12\varepsilon + 4\rho(3\delta + 10\varepsilon)$, by Lemma 5-5. ∎

Lemma 5-7 bounds the amount of real time between the time a nonfaulty process receives a round i message from another nonfaulty process and the time the last nonfaulty process begins round i + 1.

Lemma 5-7: For any $i \geq 0$ and nonfaulty processes p and q,

$tmax^{i+1} - arr^i_p(q) \leq 5\delta + 19\varepsilon + 4\rho(3\delta + 10\varepsilon)$.

Proof: $tmax^{i+1} - arr^i_p(q) = (tmax^{i+1} - t^{i+1}_p) + (t^{i+1}_p - t^i_p) + (t^i_p - t^i_q) - (arr^i_p(q) - t^i_q)$

$\leq (\delta + 3\varepsilon) + (4\delta + 12\varepsilon + 4\rho(3\delta + 10\varepsilon)) + (\delta + 3\varepsilon) - (\delta - \varepsilon)$, by Lemmas 5-2 and 5-5 and the lower bound on the message delay

$= 5\delta + 19\varepsilon + 4\rho(3\delta + 10\varepsilon)$. ∎

The next lemma bounds the error in a nonfaulty process' estimate of another nonfaulty process' local time at a particular real time.

Lemma 5-8: Let p and r be nonfaulty. Then

$|DIFF^i_p(r) + C^i_p(tmax^{i+1}) - C^i_r(tmax^{i+1})| \leq \varepsilon + \rho(11\delta + 39\varepsilon)$.

Proof: $|DIFF^i_p(r) + C^i_p(tmax^{i+1}) - C^i_r(tmax^{i+1})|$

$= |VAL^i_p(r) + \delta - ARR^i_p(r) + C^i_p(tmax^{i+1}) - C^i_r(tmax^{i+1})|$.

If the quantity in the absolute value signs is negative, then this expression is equal to

$C^i_r(tmax^{i+1}) - C^i_p(tmax^{i+1}) + C^i_p(arr^i_p(r)) - \delta - VAL^i_p(r)$

$\leq C^i_r(tmax^{i+1}) - C^i_p(tmax^{i+1}) + C^i_p(arr^i_p(r)) - \delta - C^i_r(arr^i_p(r) - \delta - \varepsilon)$, since the delay is at most $\delta + \varepsilon$

$\leq C^i_r(tmax^{i+1}) - C^i_p(tmax^{i+1}) + C^i_p(arr^i_p(r)) - \delta - C^i_r(arr^i_p(r)) + (1 + \rho)(\delta + \varepsilon)$, since the clock drift is at most $1 + \rho$

$= (C^i_r(tmax^{i+1}) - C^i_p(tmax^{i+1})) - (C^i_r(arr^i_p(r)) - C^i_p(arr^i_p(r))) - \delta + \delta + \varepsilon + \rho\delta + \rho\varepsilon$

$\leq 2\rho(tmax^{i+1} - arr^i_p(r)) + \varepsilon + \rho\delta + \rho\varepsilon$, by Lemma 4-2

$\leq 2\rho(5\delta + 19\varepsilon) + \varepsilon + \rho\delta + \rho\varepsilon$, by Lemma 5-7

$= \varepsilon + \rho(11\delta + 39\varepsilon)$.

If the quantity in the absolute value signs is positive, a similar argument shows that $|DIFF^i_p(r) + C^i_p(tmax^{i+1}) - C^i_r(tmax^{i+1})| \leq \varepsilon + \rho(11\delta + 37\varepsilon)$. ∎

The next lemma bounds how far apart two processes' i-th clocks are at the time when the last process begins round $i + 1$. The bound is in terms of how far apart the clocks are when the last process begins round i.

Lemma 5-9: For any nonfaulty p and q, and any i,

$|C^i_p(tmax^{i+1}) - C^i_q(tmax^{i+1})| \leq B^i + 8\rho(\delta + 3\varepsilon)$.

Proof: $|C^i_p(tmax^{i+1}) - C^i_q(tmax^{i+1})|$

$\leq |C^i_p(tmax^i) - C^i_q(tmax^i)| + |(C^i_q(tmax^{i+1}) - C^i_q(tmax^{i+1})) - (C^i_p(tmax^i) - C^i_q(tmax^i))|$

$\leq B^i + 2\rho(tmax^{i+1} - tmax^i)$, by definition of B^i and Lemma 4-2

$\leq B^i + 2\rho(4\delta + 12\varepsilon)$, by Lemma 5-6 and ignoring ρ^2 terms

$= B^i + 8\rho(\delta + 3\varepsilon)$. ∎

Now we can state the main result, bounding B^{i+1} in terms of B^i.

Theorem 5-10: $B^{i+1} \leq \frac{1}{2}B^i + 2\varepsilon + 2\rho(11\delta + 39\varepsilon)$.

Proof: $B^{i+1} = \max\{|C^{i+1}_p(tmax^{i+1}) - C^{i+1}_q(tmax^{i+1})|\}$ for nonfaulty p and q.

Let $x = \varepsilon + \rho(11\delta + 39\varepsilon)$.

We now define three multisets U, V, and W that satisfy the hypotheses of Lemma A-4. Let

$U = DIFF^i_p + C^i_p(tmax^{i+1})$,

$V = DIFF^i_q + C^i_q(tmax^{i+1})$, and

$W = \{C^i_r(tmax^{i+1})$: r is nonfaulty$\}$.

U and V have size n; W has size $n - f$.

Define an injection from W to U as follows. Map each element C^i_r in W to $DIFF^i_p(r) + C^i_p(tmax^{i+1})$ in U. Since Lemma 5-8 implies that

$|DIFF^i_p(r) + C^i_p(tmax^{i+1}) - C^i_r(tmax^{i+1})| \leq x$

for all the n – f nonfaulty processes, $d_x(W,U) = 0$. Similarly, $d_x(W,V) = 0$.

By Lemma 5-9, $diam(W) \leq B^i + 8\rho(\delta + 3\varepsilon)$. Thus, Lemma A-4 implies

$|mid(reduce(U)) - mid(reduce(V))| \leq ½diam(W) + 2x$

$= ½B^i + 2\varepsilon + 2\rho(11\delta + 39\varepsilon)$.

Since $mid(reduce(U)) = mid(reduce(DIFF^i_p + C^i_p(tmax^{i+1})))$

$= mid(reduce(DIFF^i_p)) + C^i_p(tmax^{i+1})$

$= ADJ^i_p + C^i_p(tmax^{i+1})$

$= C^{i+1}_p(tmax^{i+1})$

and similarly $mid(reduce(V)) = C^{i+1}_q(tmax^{i+1})$, the result follows. ∎

We obtain an approximate bound on how closely this algorithm will synchronize the clocks by considering the limit of B^i as the round number increases without bound.

Theorem 5-11: This algorithm can synchronize clocks to within $4\varepsilon + 4\rho(11\delta + 39\varepsilon)$.

Proof: $\lim_{i\to\infty} B^i$

$= \lim_{i\to\infty}[B^0/2^i + (1 + 1/2 + ... + 1/2^{i-1})(2\varepsilon + 2\rho(11\delta + 39\varepsilon))]$

$= 4\varepsilon + 4\rho(11\delta + 39\varepsilon)$, since the limit of the geometric series is 2. ∎

As was the case for Algorithm 4-1, if the number of processes, n, increases while f, the number of faulty processes remained fixed, a greater closeness of synchronization can be achieved by modifying Algorithm 5-1 so that it computes the mean instead of the midpoint of the range of values. which approaches $2\varepsilon + 2\rho P$ as n approaches infinity.

After modifying Algorithm 5-1, we get

$B^i \leq B^{i-1}f/(n-2f) + 2\varepsilon + 2\rho(11\delta + 39\varepsilon)$.

This is the same as

$B^i \leq B^0 f/(n-2f) + (1 - (f/(n-2f))^i)/(1 - f/(n-2f))(2\varepsilon + 2\rho(11\delta + 39\varepsilon))$, which approaches $2\varepsilon + 2\rho(11\delta + 39\varepsilon)$ as n approaches infinity.

5.4 Determining the Number of Rounds

The nonfaulty processes must determine how many rounds of this algorithm must be run to establish the desired degree of synchronization before switching to the maintenance algorithm. The basic idea is for each nonfaulty process p to estimate B^0, and then calculate a sufficient number of rounds, $NROUNDS_p$, using the known rate of convergence. B^0 is estimated by having p calculate an overestimate and an underestimate for $C^0_q(tmax^0)$ for each q, and letting the estimated B^0 be the difference between the maximum overestimate and the minimum underestimate.

Let p's overestimate for $C^0_q(tmax^0)$ be $OVER_p(q)$ and p's underestimate for $C^0_q(tmax^0)$ be $UNDER_p(q)$.

For the overestimate, we assume that q's clock is fast, and that the maximum amount of time elapses between t^0_q (when q sent the message) and $tmax^0$. That maximum is $\delta + \varepsilon$ since every nonfaulty process begins round 0 as soon as it receives a message. Thus,

$$OVER_p(q) = VAL^0_p(q) + (1 + \rho)(\delta + \varepsilon).$$

Similarly, we can derive the underestimate. We assume that q is the last nonfaulty process to begin round 0. Thus,

$$UNDER_p(q) = VAL^0_p(q).$$

Process p computes its estimate of B^0,

$$B^0_p = \max_q\{OVER_p(q)\} - \min_q\{UNDER_p(q)\}.$$

Now p estimates how many rounds are needed until the spread is close enough. There is a predetermined $\gamma \geq 4\varepsilon + 4\rho(11\delta + 39\varepsilon)$, which is the desired closeness of synchronization for the start-up algorithm. After j rounds,

$$B^j \leq B^0_p/2^j + (1 + 1/2 + \ldots + 1/2^{j-1})(2\varepsilon + 2\rho(11\delta + 39\varepsilon)).$$

Process p sets the right hand side equal to γ and solves for j to obtain its estimate of the required number of rounds, $NROUNDS_p$.

Now each process executes a Byzantine Agreement protocol on the vector of NROUNDS values, one value for each process. The processes are guaranteed to have the same vector at the end of the Byzantine Agreement protocol. Each process chooses the $(f+1)$-st smallest element of the resulting vector as the required number of rounds. The smallest number of rounds computed by a nonfaulty process will suffice to achieve the desired closeness of synchronization. Variations in the number of rounds computed by different nonfaulty processes are due to spurious values introduced by faulty processes and to different message delays. However, the range computed by any nonfaulty process is guaranteed to include the actual values of all nonfaulty processes at $tmax^0$, so the range determined by the process that computes the smallest number of rounds also includes all the actual values. In order to guarantee that each process chooses a number of rounds that is at least as large as the smallest one computed by a nonfaulty process, it chooses the $(f+1)$-st smallest element of the vector of values.

Any Byzantine Agreement protocol requires at least $f+1$ rounds. The processes can execute this algorithm in parallel with the clock synchronization algorithm, beginning at round 0. The clock synchronization algorithm imposes a round structure on the processes' communications. The Byzantine Agreement algorithm can be executed using this round structure. Each BA message can also include information needed for the clock synchronization algorithm (namely, the current clock value). However, the processes will always need to do at least $f+2$ rounds, one to obtain the estimated number of rounds and $f+1$ for the Byzantine Agreement algorithm.

5.5 Switching to the Maintenance Algorithm

After the processes have done the required number of rounds (denoted by r throughout this section) of the start-up algorithm, they cease executing it. The processes should begin the maintenance algorithm as soon as possible after ending the start-up algorithm in order to minimize the inaccuracy introduced by the clock drift.

In the maintenance algorithm each process broadcasts its clock value when its clock reaches T^i, for $i = 0, 1, \ldots$, where $T^{i+1} = T^i + P$. Let T^0 be a multiple of P. It is shown below in Lemma 5-13 that the first multiple of P reached by nonfaulty p's clock after finishing the required r rounds differs by at most one from the first multiple reached by nonfaulty q's clock after the r rounds. When a process reaches the first multiple of P after it has ended the start-up algorithm, it broadcasts its clock value as in the maintenance algorithm, but doesn't update its clock. At the next multiple of P, the process begins the full maintenance algorithm by broadcasting its clock value and updating its clock. (It will receive clock values from all nonfaulty processes.)

The analysis introduces a new quantity, β_1, representing an upper bound on the closeness of the nonfaulty processes' clocks at $tmax^r$. That is, for any nonfaulty processes p and q, $|C^r_p(tmax^r) - C^r_q(tmax^r)| \leq \beta_1$. We show that if the following five inequalities are satisfied by the parameters, then the switch from the start-up algorithm to the maintenance algorithm (with parameter β) can be accomplished.

(1) $\beta_1 > 4\varepsilon + 4\rho(11\delta + 39\varepsilon)$

(2) $\beta \geq (\beta_1 + 2\varepsilon + \rho(6P - \beta_1 + 2\delta + 12\varepsilon)) / (1 - 8\rho)$

(3) $P > 2(1 + \rho)(\beta + \varepsilon) + (1 + \rho)\max\{\delta, \beta + \varepsilon\} + \rho\delta$

(4) $P \leq \beta/4\rho - \varepsilon/\rho - \rho(\beta + \delta + \varepsilon) - 2\beta - \delta - 2\varepsilon$

(5) $\beta \geq 4\varepsilon + 4\rho(3\beta + \delta + 3\varepsilon) + 8\rho^2(\beta + \delta + \varepsilon)$

The first inequality is imposed by the limitation on how closely the start-up algorithm can synchronize. The second inequality reflects the inaccuracy introduced during the switch. The last three are simply repeated from Section 4.5.1.

First we show that β_1 can be attained by the start-up algorithm.

Lemma 5-12: There exists an integer i such that $B^i \leq \beta_1$.

Proof: Since β_1 must be larger than $4\varepsilon + 4\rho(11\delta + 39\varepsilon)$, the result follows from Theorem 5-11, which states that the closeness of synchronization approaches $4\varepsilon + 4\rho(11\delta + 39\varepsilon)$ as the round number, i, increases. ∎

Note that the number of rounds, r, that the processes agree on is $\geq$ i, and that the worst-case B^r is no more than the worst-case B^i, which is at most β_1.

Lemma 5-13 shows that the first multiple of P reached by a nonfaulty process after finishing the start-up algorithm differs by at most one from that reached by another nonfaulty process.

Lemma 5-13: Let p and q be nonfaulty processes. Then

$|C^r_q(t^r_q) - C^r_p(t^r_p)| \leq P.$

Proof: $|C^r_q(t^r_q) - C^r_p(t^r_p)| \leq |C^r_q(t^r_p) + (1 + \rho)(t^r_q - t^r_p) - C^r_p(t^r_p)|$

$\leq |C^r_q(t^r_p) - C^r_p(t^r_p)| + (1 + \rho)(\delta + 3\varepsilon)$, by Lemma 5-2

$\leq |(C^r_q(t^r_p) - C^r_q(tmax^r)) - (C^r_p(t^r_p) - C^r_p(tmax^r))| + |C^r_q(tmax^r) - C^r_p(tmax^r)|$

$+ (1 + \rho)(\delta + 3\varepsilon)$

$\leq 2\rho(tmax^r - t^r_p) + \beta_1 + (1 + \rho)(\delta + 3\varepsilon)$, by Lemma 4-2 and definition of β_1

$\leq 2\rho(\delta + 3\varepsilon) + \beta_1 + (1 + \rho)(\delta + 3\varepsilon)$, by Lemma 5-2

$= \beta_1 + (1 + 3\rho)(\delta + 3\varepsilon)$.

Suppose in contradiction that $P < \beta_1 + (1 + 3\rho)(\delta + 3\varepsilon)$. By solving inequality (2) for β_1, we get

$\beta_1 \leq (\beta - 2\varepsilon - \rho(8\beta + 2\delta + 12\varepsilon + 6P)) / (1 - \rho)$,

which implies that

$P < (\beta - 2\varepsilon - \rho(8\beta + 2\delta + 12\varepsilon + 6P)) / (1 - \rho) + (1 + 3\rho)(\delta + 3\varepsilon)$.

This simplifies to $P < (\beta + \delta + \varepsilon - 8\rho\beta + \rho\delta - 3\rho\varepsilon) / (1 + 5\rho)$.

Combining this with inequality (3) yields

$2(1 + \rho)(\beta + \varepsilon) + (1 + \rho)\delta + \rho\delta < P < (\beta + \delta + \varepsilon - 8\rho\beta + \rho\delta - 3\rho\varepsilon) / (1 + 5\rho)$.

Solving for β gives $\beta < -(\varepsilon + 6\rho\delta + 15\rho\varepsilon) / (1 + 20\rho)$, which is a contradiction. ∎

The rest of the section is devoted to showing that the difference in real times when nonfaulty processes' clocks reach the first multiple of P at which they will all perform the maintenance algorithm is less than or equal to β. Consequently, this β can be preserved by the maintenance algorithm.

Define kP to be the first multiple of P reached by any nonfaulty process' r-th clock. The first multiple of P reached by any other nonfaulty process is either kP or (k+1)P, by Lemma 5-13. At (k+1)P some of the nonfaulty processes will actually update their clocks, and at (k+2)P all of them will update their clocks.

Recall that $(k+1)P = T^{k+1}$ and $U^{k+1} = T^{k+1} + (1 + \rho)(\beta + \delta + \varepsilon)$. Let $u^{k+1}_p = c^r_p(U^{k+1})$ and similarly for q.

Let s and t be two nonfaulty processes. Here is a description of the worst case:

- s has the smallest clock value at $tmax^r$, barely above $(k-1)P$, and its clock is slow.
- t's clock is fast and is β_1 ahead of s's at $tmax^r$.
- s updates its clock at U^{k+1}, by decrementing it as much as possible.

- t updates its clock at U^{k+1}, by incrementing it as much as possible.

First we must bound how far apart in real time nonfaulty processes' r-th clocks reach U^{k+1}.

Lemma 5-14: Let p and q be nonfaulty processes. Then

$|c^r_p(U^{k+1}) - c^r_q(U^{k+1})| \leq (1-\rho)\beta_1 + 2\rho(2P + \beta + \delta + \varepsilon)$.

Proof: Without loss of generality, suppose $c^r_p(U^{k+1}) \geq c^r_q(U^{k+1})$. Then

$|c^r_p(U^{k+1}) - c^r_q(U^{k+1})| = c^r_p(U^{k+1}) - c^r_q(U^{k+1})$

$= (c^r_p(U^{k+1}) - tmax^r) - (c^r_q(U^{k+1}) - tmax^r)$

$\leq (C^r_p(u^{k+1}_p) - C^r_p(tmax^r))(1+\rho) - (C^r_q(u^{k+1}_q) - C^r_q(tmax^r))(1-\rho)$, by the bounds on the drift rate

$\leq (2P + (1+\rho)(\beta + \delta + \varepsilon))(1+\rho) - (2P + (1+\rho)(\beta + \delta + \varepsilon) - \beta_1)(1-\rho)$

$= (1-\rho)\beta_1 + 2\rho(2P + \beta + \delta + \varepsilon)$. ∎

Next, we bound the additional spread introduced by the resetting of the clocks.

Lemma 5-15: Let s and t be the nonfaulty processes described above. Then

(a) $c^{r+1}_s(U^{k+1}) - c^r_s(U^{k+1}) \leq (1+\rho)(\varepsilon + \rho(4\beta + \delta + 5\varepsilon))$, and

(b) $c^r_t(U^{k+1}) - c^{r+1}_t(U^{k+1}) \leq (1+\rho)(\varepsilon + \rho(4\beta + \delta + 5\varepsilon))$.

Proof: (a) By Lemma 4-15, we know that s's new clock is at most $\alpha = \varepsilon + \rho(4\beta + \delta + 5\varepsilon)$ less than the "smallest" of the previous nonfaulty clocks at $c^r_s(U^{k+1}) = u^{k+1}_s$. Since s had the smallest clock before, $C^{r+1}_s(u^{k+1}_s) \geq C^r_s(u^{k+1}_s) - \alpha$. By the lower bound on the drift rate,

$c^{r+1}_s(U^{k+1}) - c^r_s(U^{k+1}) \leq (1+\rho)\alpha$.

(b) Lemma 4-15 also states that t's new clock is at most α more than the "largest" of the previous nonfaulty clocks at u^{k+1}_t, which was t's clock. The argument is similar to (a). ∎

Finally, we can bound the maximum difference in real time between two nonfaulty processes' clocks reaching T^{k+2}. Let i_p be the index of p's logical clock that is in effect when T^{k+2} is reached.

Theorem 5-16: Let p and q be nonfaulty processes and $i = i_p$ and $j = i_q$. Then

$|c^i_p(T^{k+2}) - c^j_q(T^{k+2})| \leq \beta$.

Proof: Without loss of generality, suppose $c^i_p(T^{k+1}) \geq c^j_q(T^{k+2})$. Then

$|c^i_p(T^{k+1}) - c^j_q(T^{k+2})| = c^i_p(T^{k+1}) - c^j_q(T^{k+2})$

$\leq c^{r+1}{}_s(T^{k+2}) - c^{r+1}{}_t(T^{k+2})$

for nonfaulty processes s and t that behave as described above.

We know from Lemma 4-2 that

$(c^{r+1}{}_s(T^{k+2}) - c^{r+1}{}_t(T^{k+2})) - (c^{r+1}{}_s(U^{k+1}) - c^{r+1}{}_t(U^{k+1}))$

$\leq 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon))$.

Thus $c^{r+1}{}_s(T^{k+2}) - c^{r+1}{}_t(T^{k+2})$

$\leq 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon)) + c^{r+1}{}_s(U^{k+1}) - c^{r+1}{}_t(U^{k+1})$

$= 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon)) + c^{r+1}{}_s(U^{k+1}) - c^{r}{}_s(U^{k+1}) + c^{r}{}_t(U^{k+1}) - c^{r+1}{}_t(U^{k+1})$

$\quad + c^{r}{}_s(U^{k+1}) - c^{r}{}_t(U^{k+1})$

$\leq 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon)) + 2(1 + \rho)(\varepsilon + \rho(4\beta + \delta + 5\varepsilon))$

$\quad + c^{r}{}_s(U^{k+1}) - c^{r}{}_t(U^{k+1})$, by Lemma 5-15

$\leq 2\rho(P - (1 + \rho)(\beta + \delta + \varepsilon)) + 2(1 + \rho)(\varepsilon + \rho(4\beta + \delta + 5\varepsilon))$

$\quad + (1 - \rho)\beta_1 + 2\rho(2P + \beta + \delta + \varepsilon)$, by Lemma 5-14

$\leq \beta$, by inequality (2). ∎

This β is approximately 6ε, which is slightly larger than the smallest one maintainable, 4ε. To shrink it back down, P can be made slightly smaller than required by the maintenance algorithm, as long as the lower bound of inequality (3) isn't violated. Since the synchronization procedure is performed more often, the clocks don't drift apart as much, and consequently, they can be more closely synchronized. Once the desired β is reached, P can be increased again. (The computational costs associated with performing the synchronization procedure and the possible degradation of validity may make it advisable to resychronize more infrequently.)

5.6 Using Only the Start-up Algorithm

A natural idea is to use Algorithm 5-1 solely, and never switch to the mainenance algorithm. Both algorithms can synchronize clocks to within approximately 4ε, so such a policy would sacrifice very little in accuracy. Using just the one algorithm is conceptually simpler and avoids introducing the additional error during the switch-over. However, if the system does no work during the period of time when processes have clocks with different indices, it is important to minimize this interval. Algorithm 5-1 has such an interval of length $\delta + 3\varepsilon$; for Algorithm 4-1, it is approximately $\beta + 2\rho(\beta + \delta + \varepsilon)$. Depending on the choice of values for the parameters, Algorithm 4-1 may be superior in this regard.

Chapter Six

Conclusion

6.1 Summary

In conclusion, we have presented a precise formal model to describe a system of distributed processes, each of which has its own clock. Within this model we proved a lower bound on how closely clocks can be synchronized even under strong simplifying assumptions.

The major part of the thesis was the description and analysis of an algorithm to synchronize the clocks of a completely connected network in the presence of clock drift, uncertainty in the message delivery time, and Byzantine process faults. Since it does not use digital signatures, the algorithm requires that more than two thirds of the processes be nonfaulty. Our algorithm is an improvement over those in [7] based on Byzantine Agreement protocols in that the number of messages per round is n^2 instead of exponential, and that the size of the adjustment made at each round is a small amount independent of the number of faults.

The algorithm in [5] works for a more general communication network, and, since it uses digital signatures, only requires that more than half the processes be nonfaulty. However, the size of the adjustment depends on the number of faulty processes.

The issue of which algorithm synchronizes the the most closely is difficult to resolve because of differing assumptions about the underlying model. For instance, Algorithm 4-1 of this thesis can achieve a closeness of synchronization of approximately 4ε in our notation. However, we assume that local processing time is negligible; otherwise Lamport [8] claims that actually there is an implicit factor of n in the ε, in which case the closeness of synchronization achieved by our algorithm depends on the number of processes as do those in [7].

We also modified Algorithm 4-1 to produce an algorithm to establish synchronization initially among clocks with arbitrary values. This algorithm also handles clock drift, uncertainty in the message delivery time, and Byzantine process faults. This problem, as far as we know, had not been addressed previously for real-time clocks.

6.2 Open Questions

It would be interesting to know more lower bounds on the closeness of synchronization achievable. For example, a question posed by J. Halpern is to determine a lower bound when the communication network has an arbitrary configuration and the uncertainty in the message delivery time is different for each link.

There are also no known lower bounds for the case of clock drift and faulty processes.

The validity of algorithm 5-1 has not been computed. If this algorithm were used solely, knowing how the processes' clocks increase in relation to real time would be of interest. Lower bounds in general for the validity conditions are not known.

It seems reasonable that there is a tradeoff between the closeness of synchronization and the validity, since the synchronization procedure must be performed more often in order to synchronize more closely, but each resychronization event potentially worsens the validity. This tradeoff has not been quantified.

M. Fischer [4] has suggested an "asynchronous" version of Algorithm 5-1 to establish synchronization. In his version, a nonfaulty process wakes up at an arbitrary time with arbitrary values for its correction variable and array of differences. Every P as measured on its *physical* (not logical) clock, the process performs the fault-tolerant averaging function and updates its clock. It seems that the clock values should converge, but at what rate?

What kind of algorithms that use the fault-tolerant averaging function can be used in more general communication graphs?

Another avenue of investigation is using the fault-tolerant averaging function together with the capability for authentication to see if algorithms with higher fault-tolerance than those of this thesis and better accuracy than those in [5] can be designed.

Appendix A

Multisets

This Appendix consists of definitions and lemmas concerning multisets needed for the proofs of Lemmas 4-9 and 5-10. These definitions and lemmas are analogous to some in [1].

A *multiset* U is a finite collection of real numbers in which the same number may appear more than once. The largest value in U is denoted *max(U)*, and the smallest value in U is denoted *min(U)*. The *diameter* of U, *diam(U)*, is max(U) - min(U). Let *s(U)* be the multiset obtained by deleting one occurrence of min(U), and *l(U)* be the multiset obtained by deleting one occurrence of max(U). If $|U| \geq 2f + 1$, we define *reduce(U)* to be $l^f s^f(U)$, the result of removing the f largest and f smallest elements of U.

Given two multisets U and V with $|U| \leq |V|$, consider an injection c mapping U to V. For any nonnegative real number x, define $S_x(c)$ to be $\{u \in U: |u - c(u)| > x\}$. We define the *x-distance* between U and V to be $d_x(U,V) = \min_c\{|S_x(c)|\}$. We say c *witnesses* $d_x(U,V)$ if $|S_x(c)| = d_x(U,V)$. The x-distance between U and V is the number of elements of U that cannot be matched up with an element of V which is the same to within x. If $|u - c(u)| \leq x$, then we say u and c(u) are *x-paired* by c. The *midpoint* of U, *mid(U)*, is ½[max(U) + min(U)].

For any multiset U and real number r, define U + r to be the multiset obtained by adding r to every element of U; that is, $U + r = \{u + r: u \in U\}$. It is obvious that mid and reduce are invariant under this operation.

The next lemma bounds the diameter of a reduced multiset.

Lemma A-1: Let U and W be multisets such that $|U| = n$, $|W| = n - f$, and $d_x(W,U) = 0$, where $n \geq 2f + 1$. Then

max(reduce(U)) ≤ max(W) + x and min(reduce(U)) ≥ min(W) - x.

Proof: We show the result for max; a similar argument holds for min. Let c witness $d_x(W,U)$. Suppose none of the f elements deleted from the high end of U are x-paired with elements of W by c. Since $d_x(W,U) = 0$, the remaining n - f elements of U are x-paired with elements of W by c, and thus every element of reduce(U) is x-paired with an element of W. Suppose max(reduce(U)) is x-paired with w in W by c. Then max(reduce(U)) ≤ w + x ≤ max(W) + x.

Now suppose one of the elements deleted from the high end of U is x-paired with an element of W by c. Let u be the largest such, and suppose it was paired with w in W. Then max(reduce(U)) $\leq$ u $\leq$ w + x $\leq$ max(W) + x. ∎

We show that the x-distance between two multisets is not increased by removing the largest (or smallest) element from each.

Lemma A-2: Let U and V be multisets, each with at least one element. Then d_x(l(U),l(V)) $\leq$ d_x(U,V) and d_x(s(U),s(V)) $\leq$ d_x(U,V).

Proof: We give the proof in detail for l; a symmetric argument holds for s. Let M = l(U) and N = l(V). Let c witness d_x(U,V). We construct an injection c' from M to N and show that $|S_x(c')| \leq |S_x(c)|$. Since $d_x(M,N) \leq |S_x(c')|$ and $|S_x(c)| = d_x(U,V)$, it follows that $d_x(M,N) \leq d_x(U,V)$.

Suppose u = max(U) and v = max(V). (These are the deleted elements.)

Case 1: c(u) = v. Define c'(m) = c(m) for all m in M. Obviously c' is an injection. $|S_x(c')| \leq |S_x(c)|$ since either $S_x(c') = S_x(c)$ or $S_x(c') = S_x(c) - \{u\}$.

Case 2: c(u) ≠ v and there is no u' in U such that c(u') = v. This is the same as Case 1.

Case 3: c(u) ≠ v, and there is u' in U such that c(u') = v. Suppose c(u) = v'. Define c'(u') = v' and c'(m) = c(m) for all m in M besides v'. Obviously c' is an injection. Now we show that $|S_x(c')| \leq |S_x(c)|$.

If u or u' or both are in $S_x(c)$ then whether or not u' is in $S_x(c')$ the inequality holds. The only trouble arises if u and u' are both not in $S_x(c)$ but u' is in $S_x(c')$. Suppose that is the case. Then |u' − c'(u')| = |u' − v'| > x. There are two possibilities:

(i) u' > v' + x. Since u is not in $S_x(c)$, |u − c(u)| = |u − v'| $\leq$ x. So v' $\geq$ u − x. Hence u' > v' + x $\geq$ u − x + x, which implies that u' > u. But this contradicts u being the largest element of U.

(ii) v' > u' + x. Since u' is not in $S_x(c)$, |u' − c(u')| = |u' − v| $\leq$ x. So u' $\geq$ v − x. Hence v' > u' + x $\geq$ v − x + x, which implies that v' > v. But this contradicts v being the largest element of V.

∎

The next lemma shows that the results of reducing two multisets, each of whose x-distance from a third multiset is 0, can't contain values that are too far apart.

Lemma A-3: Let U, V, and W be multisets such that |U| = |V| = n and |W| = n − f, where n > 3f. If d_x(W,U) = 0 and d_x(W,V) = 0, then

min(reduce(U)) − max(reduce(V)) $\leq$ 2x.

Proof: First we show that $d_{2x}(U,V) \leq f$. Let c_U witness d_x(W,U) and c_V witness d_x(W,V). Define an injection c from U to V as follows: if there is w in W such that $c_U(w)$ = u, then let c(u) = $c_V(w)$; otherwise, let c(u) be any unused element of V. For each of the $n - f$ elements w in W, there is u in U such that $u = c_U(w)$. Thus $|u - c(u)| \leq |u - w| + |w - c(u)| = |c_U(w) - w| + |w - c_V(w)| \leq x + x = 2x$. Thus $S_{2x}(c) \leq f$, so $d_{2x}(U,V) \leq f$.

Then by applying Lemma A-2 f times, we know that $d_{2x}(\text{reduce}(U),\text{reduce}(V)) \leq f$. Since $|\text{reduce}(U)| = |\text{reduce}(V)| = n - 2f > f$, there are u in reduce(U) and v in reduce(V) such that $|u - v| \leq 2x$. Thus $\min(\text{reduce}(U)) - \max(\text{reduce}(V)) \leq u - v \leq 2x$. ∎

Lemma A-4 is the main multiset result. It bounds the difference between the midpoints of two reduced multisets in terms of a particular third multiset.

Lemma A-4: Let U, V, and W be multisets such that $|U| = |V| = n$ and $|W| = n - f$, where $n > 3f$. If $d_x(W,U) = 0$ and $d_x(W,V) = 0$, then

$$|\text{mid}(\text{reduce}(U)) - \text{mid}(\text{reduce}(V))| \leq \tfrac{1}{2}\text{diam}(W) + 2x.$$

Proof: $|\text{mid}(\text{reduce}(U)) - \text{mid}(\text{reduce}(V))|$

$= \tfrac{1}{2}|\max(\text{reduce}(U)) + \min(\text{reduce}(U)) - \max(\text{reduce}(V)) - \min(\text{reduce}(V))|$

$= \tfrac{1}{2}|\max(\text{reduce}(U)) - \min(\text{reduce}(V)) + \min(\text{reduce}(U)) - \max(\text{reduce}(V))|$

If the quantity inside the absolute value signs is nonnegative, this expression is equal to

$\tfrac{1}{2}[\max(\text{reduce}(U)) - \min(\text{reduce}(V)) + \min(\text{reduce}(U)) - \max(\text{reduce}(V))]$

$\leq \tfrac{1}{2}(\max(W) + x - (\min(W) - x) + \min(\text{reduce}(U)) - \max(\text{reduce}(V)))$, by applying Lemma A-1 twice

$= \tfrac{1}{2}(\text{diam}(W) + 2x + \min(\text{reduce}(U)) - \max(\text{reduce}(V)))$

$\leq \tfrac{1}{2}(\text{diam}(W) + 2x + 2x)$, by Lemma A-3

$= \tfrac{1}{2}\text{diam}(W) + 2x$.

If the quantity inside the absolute value is nonpositive, then symmetric reasoning gives the result. ∎

References

[1] D. Dolev, N. Lynch, S. Pinter, E. Stark and W. Weihl.
Reaching Approximate Agreement in the Presence of Faults.
In *Proceedings of the 3rd Annual IEEE Symposium on Distributed Software and Database Systems*. 1983.

[2] D. Dolev, J. Halpern and R. Strong.
On the Possibility and Impossibility of Achieving Clock Synchronization.
In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*. 1984.

[3] C. Dwork, N. Lynch and L. Stockmeyer.
Consensus in the Presence of Partial Synchrony.
In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*. 1984.

[4] M. Fischer.
Personal communication.

[5] J. Halpern, B. Simons and R. Strong.
Fault-Tolerant Clock Synchronization.
In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*. 1984.

[6] L. Lamport.
Time, clocks, and the ordering of events in a distributed system.
Communications of the ACM 21(7), July, 1978.

[7] L. Lamport and P.M. Melliar-Smith.
Synchronizing clocks in the presence of faults.
Research Report, SRI International, March, 1982.

[8] L. Lamport.
Personal communication.

[9] K. Marzullo.
Loosely-Coupled Distributed Services: a Distributed Time Service.
PhD thesis, Stanford University, 1983.