INFORMATION AND COMPUTATION 77, 1–36 (1988)

A New Fault-Tolerant Algorithm for Clock Synchronization

JENNIFER LUNDELIUS WELCH AND NANCY LYNCH

Laboratory for Computer Science,
Massachusetts Institute of Technology,
Cambridge, Massachusetts 02139

We describe a new fault-tolerant algorithm for solving a variant of Lamport's clock synchronization problem. The algorithm is designed for a system of distributed processes that communicate by sending messages. Each process has its own read-only physical clock whose drift rate from real time is very small. By adding a value to its physical clock time, the process obtains its local time. The algorithm solves the problem of maintaining closely synchronized local times, assuming that processes' local times are closely synchronized initially. The algorithm is able to tolerate the failure of just under one-third of the participating processes. It maintains synchronization to within a small constant, whose magnitude depends upon the rate of clock drift, the message delivery time and its uncertainty, and the initial closeness of synchronization. We also give a characterization of how far the clocks drift from real time. Reintegration of a repaired process can be accomplished using a slight modification of the basic algorithm. A similar style algorithm can also be used to achieve synchronization initially.

1. INTRODUCTION

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 con-

0890-5401/88 $3.00

ditions complicate the design and analysis of algorithms. In this paper, we describe an algorithm which meets these requirements, assuming that the clocks are initially close together and that fewer than one-third of the processes are faulty.

In our model, processes are assumed to have access to local read-only physical clocks, which are subject to a very small rate of drift. A process' local time is obtained by adding the value of the physical clock to the value of a local "correction" variable. 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. All messages are delivered within a fixed amount of time plus or minus some uncertainty. We do not require the existence of unforgeable signatures.

The problem of keeping clocks synchronized in the fault-free case was first addressed by Lamport in [La]. Further work in association with Melliar–Smith [LM] produced three fault-tolerant algorithms. Our work is based on the *interactive convergence* algorithm in [LM]. Their algorithm also requires a reliable, completely connected communication network, and handles arbitrary process faults. It runs in rounds, resynchronizing every so often to correct for the clock drift. At every round, each process obtains a value for each of the other processes' clocks, and sets its clock to the average of those values that are not too different from its own.

Our algorithm also runs in rounds. Clock values are collected at each round, but they are averaged using a fault-tolerant averaging function based on those in [DLPSW1] to calculate an adjustment. 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 roughly halved at each round. Consequently, the averaging function can be considered the heart of the algorithm.

An important capability of a practical clock synchronization algorithm is to allow a failed process that has been repaired to synchronize its clock with the other nonfaulty processes. Our algorithm provides a very simple method for this reintegration.

The problem addressed so far 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. The fact that the fault-tolerant averaging function used in our maintenance algorithm halves the error at each round suggests that it could be used to bring into synchronization clocks that begin with arbitrary values. In fact, a variant of the algorithm in this paper can be used to establish the initial

synchronization, in the face of clock drift, uncertainty in the message delivery time, and arbitrary process faults. This variant is also presented.

We consider the main contributions of this paper to be the following. The new algorithm itself is of interest because it has efficient behavior, supports reintegration of repaired processes, and can be modified to establish synchronization initially. In fact, the algorithm, with slight modifications, has been implemented. Precise statements of the conditions to be achieved by the algorithm are provided, including one for validity, and the correctness of the main algorithm is carefully demonstrated. Finally, clock synchronization is shown to be an interesting application for work on approximate agreement.

The remainder of this paper is organized as follows: In Section 2 we describe the underlying model upon which our work is based. In Section 3 the assumptions we make about clock behavior are given and the problem to be solved is stated precisely, in terms of the model. Our basic algorithm is presented in Section 4; it is described first in words, and then in a high-level "programming language." Sections 5 through 8 show with what precision the algorithm solves the problem posed earlier. In Section 9 we briefly discuss extensions to the basic algorithm in order to reintegrate a repaired process and to establish synchronization initially, and discuss the implementation. Finally, in Section 10 we compare our algorithm to several others.

2. A Model for Systems of Processes with Clocks

This section is an informal description of the model used to describe a system of processes which have physical clocks. A completely formal development can be found in [Lu1].

2.1. *Processes, Clocks, and Systems*

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. Processes are interrupt-driven. An interrupt can be an ordinary message, consisting of text and the sending process' name, or initial system start-up, or the event that the process' physical clock reaches a particular value. It is convenient to model the latter two events uniformly with the messages, as special START and TIMER messages. START indicates that the recipient should begin the algorithm, and TIMER is received when the recipient's physical clock has reached a designated time. By sending a TIMER message to itself, 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.

In more detail, a *process* is 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. At a process step, the process receives a message, changes state, and sends out some messages. If the process is *nonfaulty*, the new state and messages sent obey the transition function. Otherwise the process is *faulty*. By not restricting the state changes or messages sent by faulty processes, we model arbitrary, or *Byzantine*, process faults.

We define a *clock* to be a monotonically increasing, everywhere differentiable function from $\mathbb{R}$ to $\mathbb{R}$, interpreted as being a function from real times to clock times, or vice versa. Throughout this paper, lower case letters will represent real times, and upper case letters clock times. A clock from real times to clock times will be denoted with upper case, and its inverse will be denoted by the same name in lower case. A *system* consists of a set of processes and a set of clocks, the *physical clocks*, from real times to clock times, one clock for each process. The physical clock for process p will be denoted Ph_p.

2.2. *The Message System*

Every process can communicate directly with every process, including itself. The *message system* is modelled by a global message buffer. When a process sends a message at real time t to another process, the message is placed in the message buffer together with a time t' greater than t. At real time t', the message is received by the proper recipient and is deleted from the buffer. The *message delay* is $t' - t$. In its initial state the message buffer contains no messages except for START messages, exactly one for each process, together with their scheduled delivery times.

When a process p sets a timer, say for time T, a TIMER message with recipient p and delivery time $\mathrm{Ph}_p^{-1}(T)$ is placed in the message buffer, provided $\mathrm{Ph}_p^{-1}(T)$ is greater than the current real time. If it is not, no message is placed in the buffer.

2.3. *Executions*

There is only one type of *event* in this model, receive(m, p), the receipt of message m by process p. In order to discuss how an event affects the system as a whole, we define a *configuration* to consist of a state for each process and a state for the message buffer. An event surrounded by two configurations of the system, e.g., (F, e, F'), is an *action*.

We define an *execution* of the system to be a mapping from real times to finite sequences of actions with the following properties:

1. Only a finite number of actions occurs before any fixed real time (so concatenating the sequences for real times in order of increasing times produces a sequence of actions);

2. the first configuration of the first action consists of initial states for the processes and the message buffer;

3. the configurations match up corectly; that is, the second configuration of an action is the same as the first one of the following action;

4. all TIMER messages received by a particular process p that arrive at real time t are ordered after any non-TIMER messages for p that arrive at real time t (so messages that arrive at the same time as a timer is due to go off get in "just under the wire");

5. an action $(F, \text{receive}(m, p), F')$ occurs at real time t if and only if m, with delivery time t, is in the message buffer in configuration F;

6. if an action $(F, \text{receive}(m, p), F')$ occurs at real time t, then the only differences between F and F' are that p's state may change and that the message buffer in F' no longer contains m but may contain some mesages and timers from p; furthermore, if p is nonfaulty, then its new state and the additions to the message buffer are determined by p's transition function acting on p's state in F, the message m, and the physical clock time $\text{Ph}_p(t)$.

The sixth property implies that at each event $\text{receive}(m, p)$, only the message buffer and process p's state are allowed to change. However, this condition is not very restrictive, since faulty processes are not constrained to obey their transition functions–they can change state arbitrarily and set whatever timers they like for themselves. Therefore, they can choose when they take steps and can do anything they want at a step.

The configuration *at time t* is the second configuration in the final action of the sequence to which t is mapped. If t is mapped to the empty sequence, then the configuration at time t is the configuration at time t', where t' is the greatest time less than t such that t' is not mapped to the empty sequence. (If t is earlier than the first action of the execution, then the configuration at time t is the first configuration of the first action of the execution.) Thus the state of a process or the message buffer at time t is determined by the configuration at time t.

3. The Clock Synchronization Problem

3.1. *ρ-Bounded Clocks*

The notion of a ρ-bounded clock is useful for defining a clock whose

drift is small, i.e., one that does not gain or lose time drastically. The amount of clock time elapsed on a ρ-bounded clock during some real time interval is close to the amount of real time that has elapsed.

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

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

A simple corollary is that $1-\rho \leqslant dC(t)/dt \leqslant 1/(1-\rho)$. The two lower bounds, $1/(1+\rho)$ and $1-\rho$, are very close for small ρ (similarly the upper bounds), and we will use whichever formulation is more convenient. Similarly, clock c is *ρ-bounded* provided that for all T

$$1/(1+\rho) \leqslant dc(T)/dT \leqslant 1+\rho$$

We give several straightforward lemmas about the behavior of ρ-bounded clocks. First we observe that the inverse of a ρ-bounded clock is itself a ρ-bounded clock, by the definition. The next three lemmas quantify changes in clock time values in terms of changes in real time values. They are all stated in terms of upper case clocks, but the previous observation implies that analogous results hold for lower case clocks (the inverses).

LEMMA 1. *Let C be any clock. If $t_1 \leqslant t_2$, then*

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

Proof. Straightforward by the mean value theorem. ∎

LEMMA 2. *Let C and D be clocks. Then for any t_1 and t_2,*

(a) $|(C(t_2) - t_2) - (C(t_1) - t_1)| \leqslant \rho\,|t_2 - t_1|$.

(b) $|(C(t_2) - D(t_2)) - (C(t_1) - D(t_1))| \leqslant 2\rho\,|t_2 - t_1|$.

Proof. (a) Note that $|(C(t_2) - t_2) - (C(t_1) - t_1)| = |(C(t_2) - C(t_1)) - (t_2 - t_1)|$. We do one of the four cases. Suppose $t_2 \geqslant t_1$ and $C(t_2) - C(t_1) \geqslant t_2 - t_1$. Then

$$\begin{aligned} |(C(t_2) - C(t_1)) - (t_2 - t_1)| &= (C(t_2) - C(t_1)) - (t_2 - t_1) \\ &\leqslant (1+\rho)(t_2 - t_1) - (t_2 - t_1), \qquad \text{by Lemma 1} \\ &= \rho\,|t_2 - t_1|. \end{aligned}$$

(b) And

$$\begin{aligned}
&|(C(t_2)-D(t_2))-(C(t_1)-D(t_1))| \\
&\quad = |((C(t_2)-t_2)-(C(t_1)-t_1))-((D(t_2)-t_2)-(D(t_1)-t_1))| \\
&\quad \leqslant |(C(t_2)-t_2)-(C(t_1)-t_1)|+|(D(t_2)-t_2)-(D(t_1)-t_1)| \\
&\quad \leqslant 2\rho\,|t_2-t_1|, \text{ by part (a).} \quad \blacksquare
\end{aligned}$$

LEMMA 3. *Let C and D be clocks, $T_1 \leqslant T_2$. Assume $|c(T)-d(T)| \leqslant \alpha$ for all T, $T_1 \leqslant T \leqslant T_2$. Let $t_1 = \min\{c(T_1), d(T_1)\}$ and $t_2 = \max\{c(T_2), d(T_2)\}$. Then for all t, $t_1 \leqslant t \leqslant t_2$,*

$$|C(t)-D(t)| \leqslant (1+\rho)\,\alpha.$$

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

Case 1. $c(T_1) \leqslant t \leqslant c(T_2)$. *Let* $T_3 = C(t)$, *so that* $T_1 \leqslant T_3 \leqslant T_2$. Then

$$\begin{aligned}
|C(t)-D(t)| &= |T_3-D(t)| \\
&\leqslant (1+\rho)|d(T_3)-t|, && \text{by Lemma 1} \\
&= (1+\rho)|d(T_3)-c(T_3)| \\
&\leqslant (1+\rho)\,\alpha, && \text{by hypothesis.}
\end{aligned}$$

Case 2. $d(T_1) \leqslant t \leqslant d(T_2)$. 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

$$\begin{aligned}
|C(t)-D(t)| = C(t)-D(t) &= (C(t)-T_1)+(T_1-D(t)) \\
&\leqslant (1+\rho)(t-c(T_1))+(1+\rho)(d(T_1)-t), && \text{by Lemma 1,} \\
&= (1+\rho)(d(T_1)-c(T_1)) \leqslant (1+\rho)\,\alpha.
\end{aligned}$$

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

3.2. *Problem Statement*

We assume that each process p has 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 $\text{CORR}_p(t)$, giving the value of p's variable CORR at real time t.

For a particular execution, we define the *local time* for p to be function L_p, which is given by $\mathrm{Ph}_p + \mathrm{CORR}_p$.

A *logical clock* of p is Ph_p plus the value of CORR_p at some time. Let C_p^0 denote the *initial logical clock* of p, given by Ph_p plus the value, in p's initial state, of CORR_p. In keeping with our notational convention, we let c_p^0 denote the inverse function of C_p^0. Each time p adjusts its CORR variable, it can be thought of as changing to a new logical clock. The local time can be thought of as a piecewise continuous function, each of whose pieces is part of a logical clock.

We make the following assumptions:

(A1) Fix a small constant ρ for the remainder of the paper. All clocks are ρ-bounded, including those of faulty processes. (Since faulty processes are permitted to take arbitrary steps, faulty clocks would not increase their power to affect the behavior of nonfaulty processes.)

(A2) 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 $3f+1$. (Dolev, Halpern, and Strong [DHS] show that it is impossible without authentication to synchronize clocks unless more than two-thirds of the processes are nonfaulty.)

(A3) Fix nonnegative constants δ and ε, with $\delta > \varepsilon$, for the remainder of the paper. The delay for every message is in the range $[\delta - \varepsilon, \delta + \varepsilon]$.

(A4) Fix a constant β and a clock time T^0 for the remainder of the paper. We want to assume that the initial logical clocks are closely synchronized, but for technical reasons, we instead give a condition on the inverses of the initial logical clocks. More formally, $|c_p^0(T^0) - c_q^0(T^0)| \leqslant \beta$, for all nonfaulty p and q. Furthermore, a START message arives at each process p at time T^0 on its initial logical clock C_p^0. These two conditions say that all the nonfaulty processes wake up within an interval of length β, when their logical clocks reach T^0.

We denote the real time $c_p^0(T^0)$ by t_p^0. We let $\mathrm{tmax}^0 = \max_{p \text{ nonfaulty}}\{t_p^0\}$ and analogously for tmin^0. These are respectively the latest and earliest real times when START messages arrive at nonfaulty processes.

The object is to design an algorithm such that every execution satisfies the following two properties:

(1) γ-agreement: $|L_p(t) - L_q(t)| \leqslant \gamma$, for all $t \geqslant \mathrm{tmin}^0$ and all nonfaulty p, q.

(2) $(\alpha_1, \alpha_2, \alpha_3)$-validity: $\alpha_1(t - \mathrm{tmax}^0) - \alpha_3 \leqslant L_p(t) - T^0 \leqslant \alpha_2(t - \mathrm{tmin}^0) + \alpha_3$, for all $t \geqslant t_p^0$ and all nonfaulty p.

The agreement property states that at any real time, all the nonfaulty

processes' clocks differ by at most γ. We would, of course, like to minimize γ in order for the clock values to be close to each other. The validity property implies that the local time of a nonfaulty process increases in some linear relation to real time. This condition can be formulated in a variety of ways; this particular one, with tmax^0 and tmin^0, fits in best with our analysis. We would like $L_p(t) - T^0$ to be close to $t - t_p^0$ (i.e., the amount of elapsed clock time to be close to the amount of elapsed real time); this is the case when α_1 and α_2 are near 1 and α_3 is near 0.

4. The Algorithm

4.1. *General Description*

The algorithm executes in a series of rounds, with a resynchronization occurring in each round at a fixed local time. The ith round for process p is triggered by its ith logical clock reaching some value T^i. (It will be shown that all the ith logical clocks of nonfaulty processes reach this value within real time β of each other.) When p's ith 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 p receives T^i messages from all the 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 its $(i+1)$st logical clock.

The process p then waits until its $(i+1)$st logical clock reaches time $T^{i+1} = T^i + P$, by setting a timer, and repeats the procedure. P, then, is the length of a round in local time. (Section 5.1 discusses constraints on the size of P and β.)

The fault-tolerant averaging function is derived from those used in [DLPSW1] 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. It turns out that using the midpoint causes the error to be halved at each round.

It is possible for the clock to be set backwards in this algorithm. However, this is not a real problem, since there are known techniques for stretching a negative adjustment out over the resynchronization interval.

4.2. *Code for an Arbitrary Process*

We use the following conventions to present algorithms for our interrupt-driven model. Several clusters of pseudo-code are listed, each one preceded by the condition under which it is to be executed. The condition must contain a *receive* statement, meaning that the receipt of a certain message triggers the execution of the cluster. The cluster manipulates the state of the process, represented by local variables, and may include the sending of messages. Since we are only concerned with deterministic algorithms, for each message received at any point in any execution, at most one cluster can be applicable. (If no cluster is applicable, then nothing happens.)

We now present our clock synchronization algorithm.

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

Local variables:

- ADJ, AV: initially arbitrary; scalars used in calculating the correction.
- ARR$[1 \cdots n]$: initially arbitrary; array containing the arrival times, measured in local time, of the most recent messages, one entry for each process.
- CORR: initially whatever value is needed to attain required degree of synchronization with other processes' clocks; scalar holding correction value for physical clock.
- FLAG: initially BCAST; flag that toggles between the values BCAST and UPDATE, used to alternate between broadcasting clock value and updating clock, upon receipt of a timer.
- T: initially T^0; scalar taking on the values T^0, T^0+P, T^0+2P, etc., the beginnings of the rounds.

Subroutines called:

- broadcast(m): send the message m to every process.
- mid(M): applied to a multiset M of real numbers, returns the midpoint of the set of values in the multiset. (The midpoint is the arithmetic mean of the smallest and largest elements in the multiset.)
- local-time( ): returns the local time (current value of the physical clock + CORR).
- reduce(A): applied to an array A, returns the multiset consisting of the elements of the array, with the f largest and f smallest elements removed.
- set-timer(T): given a time value T, sets a timer for when the physical clock reaches the value $T-$CORR, for the current value of CORR. (Equivalent to setting a timer for when the current logical clock reaches T.)

Code:

receive(m) from q:
ARR[q] := local-time( )
(receive(START) or receive(TIMER)) and (FLAG = BCAST):
broadcast(T)
set-timer($T + (1+\rho)(\beta+\delta+\varepsilon)$)
FLAG := UPDATE
receive(TIMER) and (FLAG = UPDATE):
AV := mid(reduce(ARR))
ADJ := $T + \delta -$ AV
CORR := CORR + ADJ
$T := T + P$
set-timer(T)
FLAG := BCAST

If an ordinary message arrives, its arrival time is stored in the array. If a timer (or start) arrives and it is time to broadcast, then the time T, marking the beginning of the current round, is broadcast, a timer is set to update the clock later on, and the flag is toggled. If a timer arrives and it is time to update the clock, then the fault-tolerant averaging function is computed, T is updated to be the beginning of the next round, and a timer is set for that time (on the new logical clock). Again, the flag is toggled.

5. Preliminaries to the Analysis

Although the algorithm is fairly simple, its analysis is surprisingly complicated and requires a long series of lemmas. Our final goal is to provide values for γ, α_1, α_2, and α_3 in the γ-agreement and $(\alpha_1, \alpha_2, \alpha_3)$-validity properties defined in Section 3.2. In this section, the notation used to analyze the algorithm is described, and bounds are given on some of the parameters. Several important facts are proved inductively for every round in Section 6. Section 7 consists of the proof of agreement, and validity is demonstrated in Section 8.

5.1. *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 when its ith logical clock C_p^i reaches time T^i (recall this is real time t_p^i). Then it sets a timer to go off when its ith logical clock reaches U^i. Define u_p^i to be the real time when this occurs. When the ith logical clock reaches U^i (at real time u_p^i), the process resets its CORR variable, thereby switching to a new logical clock, denoted

C_p^{i+1}. Also at real time u_p^i, the process sets a timer for when its new logical clock C_p^{i+1} reaches T^{i+1}. We will require that P be big enough to ensure that C_p^{i+1} has not already passed T^{i+1}, and therefore this new timer is being set for a time in the future. The process moves through an infinite sequence of clocks C_p^0, C_p^1, etc., where C_p^0 is in force in the interval of real time $(-\infty, u_p^0)$, and each C_p^i, $i \geq 1$, is in force in the interval of real time $[u_p^{i-1}, u_p^i)$.

The interval of real time $[t_p^i, t_p^{i+1})$ constitutes *round i* for process p.

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

For p and q nonfaulty, let $\text{ARR}_p^i(q)$ denote the time of arrival of a T^i message from q to p, sent when q's ith logical clock reaches time T^i, where the arrival time is measured on p's logical clock C_p^i. (We will prove that C_p^i is actually in force by the time this message arrives.) ARR_p^i will denote the multiset of values $\text{ARR}_p^i(q)$ for all q. Let AV_p^i denote the value of AV calculated by p using the ARR_p^i values, i.e., the "average" arrival time of T^i messages calculated by throwing out the f earliest and f latest ones and taking the midpoint of the rest. Let ADJ_p^i denote the corresponding value of ADJ calculated by p, i.e., the "adjustment" calculated by subtracting the average from the ideal arrival time. Thus, $C_p^{i+1} = C_p^i + \text{ADJ}_p^i$.

5.2. *Bounds on the Parameters*

In a real system, the parameters ρ (drift rate), δ (median message delay), and ε (uncertainty in the delay) would be fixed by the hardware and low-level communication protocols employed, whereas the algorithm designer would have some freedom in the choice of P (round length) and β (how closely in real time processes reach the same round), subject to the reasonableness of the assumption that the clocks initially begin the algorithm within β. To keep the clocks as closely synchronized as possible, β must be as small as possible. However, the smaller β is, the smaller P must be (i.e., the more frequently we must synchronize).

However, P cannot be arbitrarily small. In order for the algorithm to work correctly, P must be 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 p at round i, which will be used to set the $(i+1)$st logical clock, arrives at a nonfaulty process q after q has already set its ith logical clock.

Although it is not obvious at this point, the analysis to be presented demonstrates that sufficient conditions relating the parameters are

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

and

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

It follows that

$$\beta \geqslant 4\varepsilon + 4\rho(4\beta + \delta + 4\varepsilon + \max\{\delta, \beta + \varepsilon\}) + 4\rho^2(3\beta + 2\delta + 3\varepsilon + \max\{\delta, \beta + \varepsilon\}).$$

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.

6. Inductive Analysis

This section is devoted to proving the following theorem.

THEOREM 4. *Let p and q be nonfaulty processes, and $i \geqslant 0$.*

(a) *If $i \geqslant 1$, then $|\mathrm{ADJ}_p^{i-1}| \leqslant (1+\rho)(\beta+\varepsilon) + \rho\delta$.*

(b) *If $i \geqslant 1$, then $U^{i-1} + \mathrm{ADJ}_p^{i-1} < T^i$.*

(c) $|t_p^i - t_q^i| \leqslant \beta$.

(d) *If $i \geqslant 1$, then $t_p^i + \delta - \varepsilon > u_q^{i-1}$.*

This theorem states that for each i and each nonfaulty process p, (a) the adjustment that created p's ith logical clock, ADJ_p^{i-1}, is bounded in size; (b) the time to broadcast round i messages is still in the future when p's ith logical clock is started; (c) p begins round i within β real time of any other nonfaulty process; and (d) p's round i message arrives at q after q has already set its ith logical clock. (Note that (b) and (d) are the same as the conditions that necessitate a lower bound on the size of P, the round length.)

The proof of the base case of the theorem, when $i = 0$, is easy: (a), (b), and (d) are vacuously true, and (c) is true by assumption A4. For the remainder of this section, we assume the theorem is true for i, for any two nonfaulty processes p and q, and prove it for $i+1$. (Thus, integer i is fixed for the rest of this section.)

6.1. *Bounding the Adjustment*

In this subsection, we prove several lemmas leading up to a proof of part (a) of Theorem 4 for $i+1$. 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 any other nonfaulty process' clock reaching T^i.

LEMMA 5. *Let p and q be nonfaulty processes. Then*

$$|(\mathrm{ARR}_p^i(q) - (T^i + \delta)) - (c_q^i(T^i) - c_p^i(T^i))| \leqslant \varepsilon + \rho(\beta + \delta + \varepsilon).$$

Proof. The intuition behind this result is as follows: p assumes that q's message took exactly δ time to arrive at p, and that its own clock has no drift. If these assumptions were true, then δ time before the message arrived would be when q reached T^i, and so $\mathrm{ARR}_p^i(q) - (T^i + \delta)$ would indeed be the diffeence in real time between the two processes' clocks reaching T^i. However, the message could have taken as much as ε more or less time, and the drift of p's clock could have introduced error the entire time between p reaching T^i and p receiving q's message, which could be as long as $\beta + \delta + \varepsilon$. A careful proof follows.

Let a be the real time of arrival of q's message at process p.

$$\begin{aligned}&|(\mathrm{ARR}_p^i(q) - (T^i + \delta)) - (c_q^i(T^i) - c_p^i(T^i))| \\ &\quad \leqslant |(\mathrm{ARR}_p^i(q) - T^i) - (a - c_p^i(T^i))| + |a - c_q^i(T^i) - \delta|.\end{aligned}$$

The second term is at most ε, by the bound on the message delay.

By applying Lemma 2 to the first term with $t_1 = a$, $t_2 = c_p^i(T^i)$, and $C = C_p^i$, we obtain an upper bound of $\rho|a - c_p^i(T^i)|$. By part (c) of Theorem 4 for i, and the bounds on the message delay, $|a - c_p^i(T^i)| \leqslant \beta + \delta + \varepsilon$. The result follows. ∎

LEMMA 6. *Let p be a nonfaulty process. Then there exist nonfaulty processes q and r with*

$$\mathrm{ARR}_p^i(q) \leqslant \mathrm{AV}_p^i \leqslant \mathrm{ARR}_p^i(r).$$

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. ∎

The following lemma proves part (a) of Theorem 4 for $i + 1$.

LEMMA 7. *For any nonfaulty process p, $|\mathrm{ADJ}_p^i| \leqslant (1 + \rho)(\beta + \varepsilon) + \rho\delta$.*

Proof. Since $\mathrm{ADJ}_p^i = T^i + \delta - \mathrm{AV}_p^i$, Lemma 6 implies that for some nonfaulty q and r,

$$T^i + \delta - \mathrm{ARR}_p^i(q) \leqslant \mathrm{ADJ}_p^i \leqslant T^i + \delta - \mathrm{ARR}_p^i(r).$$

Now Lemma 5 implies

$$|T^i + \delta - \mathrm{ARR}_p^i(r)| \leqslant |c_q^i(T^i) - c_p^i(T^i)| + \varepsilon + \rho(\beta + \varepsilon + \delta)$$
$$\leqslant \beta + \varepsilon + \rho(\beta + \varepsilon + \delta), \qquad \text{by part (c) of Theorem 4 for } i.$$

This simplifies to the required expression. ∎

6.2. *Timers Are Set in the Future*

In this subsection, we prove part (b) of Theorem 4 for $i+1$, that when a process sets its $(i+1)$st logical clock, the scheduled time for the next broadcast is still in the future. We simply assume that P, the round length, is sufficiently large.

LEMMA 8. *For any nonfaulty process p,* $U^i + \mathrm{ADJ}_p^i < T^{i+1}$.

Proof. By part (a) of Theorem 4 for $i+1$, which has already been proved,

$$U^i + \mathrm{ADJ}_p^i$$
$$\leqslant U^i + (1+\rho)(\beta + \varepsilon) + \rho\delta$$
$$= U^i + (2(1+\rho)(\beta+\varepsilon) + (1+\rho)\,\delta + \rho\delta) - (1+\rho)(\beta + \delta + \varepsilon)$$
$$< U^i + P - (1+\rho)(\beta + \delta + \varepsilon), \qquad \text{by the assumed lower bound on } P$$
$$= T^{i+1}, \qquad \text{by definition of } U^i \text{ and } T^{i+1}. \quad ∎$$

6.3. *Bounding the Separation of Clocks*

In this subsection, we prove part (c) of Theorem 4 for $i+1$, i.e., that nonfaulty processes' clocks reach T^{i+1} within β of each other. Several lemmas lead up to this result.

This lemma is the key to why the algorithm works. It states that the adjustments of processes p and q compensate for the difference in their clocks' reaching T^i with an error of approximately $\beta/2$. Since the clocks reached T^i within β real time, the difference between the clocks has been cut roughly in half. 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. The averaging function is defined on multisets of values, and the proof of the lemma requires some definitions and results about multisets, which are presented in the Appendix.

LEMMA 9. *Let p and q be nonfaulty processes. Then*

$$|(c_p^i(T^i) - c_q^i(T^i)) - (\mathrm{ADJ}_p^i - \mathrm{ADJ}_q^i)| \leqslant \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon).$$

Proof. We define multisets U, V, and W, and show that they satisfy the hypotheses of Lemma 24. Let

$$U = c_p^i(T^i) - (T^i + \delta) + \mathrm{ARR}_p^i,$$
$$V = c_q^i(T^i) - (T^i + \delta) + \mathrm{ARR}_q^i,$$

and

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

U and V have size n and W has size at least $n-f$. Let $x = \varepsilon + \rho(\beta + \delta + \varepsilon)$. Define an injection from W to U as follows: For all r, $c_r^i(T^i)$ in W is mapped to $c_p^i(T^i) - (T^i + \delta) + \mathrm{ARR}_p^i(r)$ in U. Since Lemma 5 implies that $|(\mathrm{ARR}_p^i(r) - (T^i + \delta)) - (c_r^i(T^i) - c_p^i(T^i))| \leqslant \varepsilon + \rho(\beta + \delta + \varepsilon)$ 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, $\mathrm{diam}(W) = \beta$. By Lemma 24, $|\mathrm{mid}(\mathrm{reduce}(U)) - \mathrm{mid}(\mathrm{reduce}(V))| \leqslant \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon)$. Since $\mathrm{mid}(\mathrm{reduce}(U)) = \mathrm{mid}(\mathrm{reduce}(c_p^i(T^i) - (T^i + \delta) + \mathrm{ARR}_p^i)) = c_p^i(T^i) - \mathrm{ADJ}_p^i$, and $\mathrm{mid}(\mathrm{reduce}(V)) = c_q^i(T^i) - \mathrm{ADJ}_q^i$, the result follows. ∎

Next we bound the distance in real time between when the new clocks of processes p and q reach T, for any T.

LEMMA 10. *Let p and q be nonfaulty processes, T any clock time. Then*

$$|c_p^{i+1}(T) - c_q^{i+1}(T)| \leqslant 2\rho\,|T - T^i| + \beta/2 + 2\varepsilon + 2\rho(2\beta + \delta + 2\varepsilon) + 2\rho^2(\beta + \delta + \varepsilon).$$

Proof. Note that $c_p^{i+1}(T) = c_p^i(T - \mathrm{ADJ}_p^i)$, and $c_q^{i+1}(T) = c_q^i(T - \mathrm{ADJ}_q^i)$. Now

$$\begin{aligned} |c_p^{i+1}(T) - c_q^{i+1}(T)| &\leqslant |c_p^i(T - \mathrm{ADJ}_p^i) - c_p^i(T^i) - (T - \mathrm{ADJ}_p^i - T^i)| \\ &\quad + |c_q^i(T - \mathrm{ADJ}_q^i) - c_q^i(T^i) - (T - \mathrm{ADJ}_q^i - T^i)| \\ &\quad + |c_p^i(T^i) - c_q^i(T^i) - (\mathrm{ADJ}_p^i - \mathrm{ADJ}_q^i)|. \end{aligned}$$

We bound the three terms separately. By Lemma 2,

$$\begin{aligned} &|c_p^i(T - \mathrm{ADJ}_p^i) - c_p^i(T^i) - (T - \mathrm{ADJ}_p^i - T^i)| \\ &\quad \leqslant \rho(|T - T^i - \mathrm{ADJ}_p^i|) \\ &\quad \leqslant \rho(|T - T^i| + (1 + \rho)(\beta + \varepsilon) + \rho\delta), \qquad \text{by Lemma 7.} \end{aligned}$$

The second term is bounded in the same way. Lemma 9 bounds the third term:

$$|(c_p^i(T^i) - c_q^i(T^i)) - (\mathrm{ADJ}_p^i - \mathrm{ADJ}_q^i)| \leqslant \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon).$$

Adding these three bounds and simplifying gives the result. ∎

The next lemma proves part (c) of Theorem 4 for $i+1$, bounding the distance between times when clocks reach T^{i+1}. This result uses the fact that we have assumed an upper bound on the round length P to ensure that the clocks cannot drift too far apart.

LEMMA 11. *For any two nonfaulty processes p and q,* $|t_p^{i+1} - t_q^{i+1}| \leqslant \beta$.

Proof. Since $t_p^{i+1} = c_p^{i+1}(T^{i+1})$, we can apply Lemma 10, so

$$|t_p^{i+1} - t_q^{i+1}| \leqslant 2\rho\,|P| + \beta/2 + 2\varepsilon + 2\rho(2\beta + \delta + 2\varepsilon) + 2\rho^2(\beta + \delta + \varepsilon).$$

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

6.4. *Bound on Message Arrival Time*

In this subsection, we prove part (d) of Theorem 4 for $i+1$, i.e., that messages arrive after the appropriate clocks have been set. This property is true because the round length P is assumed to be sufficiently large.

LEMMA 12. *For any two nonfaulty processes p and q,* $t_q^{i+1} + \delta - \varepsilon > u_p^i$.

Proof. Since $t_q^{i+1} + \delta - \varepsilon \geqslant t_p^{i+1} - \beta + \delta - \varepsilon$ by part (c) of Theorem 4 for $i+1$ (which was just proved), it suffices to show that

$$t_p^{i+1} - u_p^i > \beta - \delta + \varepsilon.$$

By Lemma 1,

$$\begin{aligned} t_p^{i+1} - u_p^i &\geqslant (C_p^{i+1}(t_p^{i+1}) - C_p^{i+1}(u_p^i))/(1+\rho) \\ &= (T^{i+1} - (U^i + \mathrm{ADJ}_p^i))/(1+\rho) \\ &\geqslant (P - (1+\rho)(\beta + \delta + \varepsilon) - \mathrm{ADJ}_p^i)/(1+\rho). \end{aligned}$$

But the lower bound on P implies that $P > 3(1+\rho)(\beta+\varepsilon) + \rho\delta$. Also, part (a) of Theorem 4 for $i+1$ shows that $\mathrm{ADJ}_p^i \leqslant (1+\rho)(\beta+\varepsilon) + \rho\delta$. Therefore,

$$\begin{aligned} t_p^{i+1} - u_p^i &> (3(1+\rho)(\beta+\varepsilon) + \rho\delta - (1+\rho)(\beta+\delta+\varepsilon) \\ &\quad - (1+\rho)(\beta+\varepsilon) - \rho\delta)/(1+\rho) \\ &= \beta - \delta + \varepsilon, \end{aligned}$$

as needed. ∎

7. Agreement

This section culminates in the main result, bounding how far apart clocks of nonfaulty processes are at any given real time.

First, we bound how closely in real time the various clocks reach corresponding values. If one is only concerned with the closeness of synchronization along the real time axis, then no further analysis is required beyond this lemma.

Lemma 13. *Let p and q be nonfaulty, $i \geqslant 0$. Then*

$$|c_p^i(T) - c_q^i(T)| \leqslant \beta + 2\rho(1+\rho)(\beta+\delta+\varepsilon)$$

for $T^{i-1} - P - (1+\rho)(\beta+\delta+\varepsilon) \leqslant T \leqslant U^i$, if $i \geqslant 1$, and for $T^0 - (1+\rho)(\beta+\delta+\varepsilon) \leqslant T \leqslant U^0$, if $i = 0$.

Proof.

Case 1. $i = 0$. Choose T with $T^0 - (1+\rho)(\beta+\delta+\varepsilon) \leqslant T \leqslant U^0$. Thus $|T - T^0| \leqslant (1+\rho)(\beta+\delta+\varepsilon)$.

$$\begin{aligned} &|c_p^0(T) - c_q^0(T)| \\ &\quad \leqslant |(c_p^0(T) - c_q^0(T)) - (c_p^0(T^0) - c_q^0(T^0))| + |c_p^0(T^0) - c_q^0(T^0)| \\ &\quad \leqslant 2\rho\,|T - T^0| + \beta, \quad \text{by Lemma 2 (for inverse clocks) and assumption A4} \\ &\quad \leqslant \beta + 2\rho(1+\rho)(\beta+\delta+\varepsilon), \quad \text{by the bound on } |T - T^0|. \end{aligned}$$

Case 2. $i > 0$. Choose T with $T^{i-1} - P - (1+\rho)(\beta+\varepsilon+\delta) \leqslant T \leqslant U^i$. Thus $|T - T^{i-1}| \leqslant P + (1+\rho)(\beta+\varepsilon+\delta)$. By Lemma 10,

$$\begin{aligned} &|c_p^i(T) - c_q^i(T)| \\ &\quad \leqslant 2\rho\,|T - T^{i-1}| + \beta/2 + 2\varepsilon + 2\rho(2\beta+\delta+2\varepsilon) + 2\rho^2(\beta+\delta+\varepsilon) \\ &\quad \leqslant 2\rho P + \beta/2 + 2\varepsilon + 2\rho(3\beta+2\delta+3\varepsilon) + 4\rho^2(\beta+\delta+\varepsilon), \\ &\qquad \text{by the bound on } |T - T^{i-1}|. \end{aligned}$$

The upper bound on P implies the result. ∎

The main effort in the next three results is to take the bound proved in Lemma 10, concerning the closeness in real times when clocks reach the same value, and to restate it in terms of the closeness of clock values at the same real time. First, we prove a bound for a nonfaulty process' $(i+1)$st clock, in terms of nonfaulty processes' ith clocks.

LEMMA 14. *Let p be nonfaulty, $i \geqslant 0$. Then there exist nonfaulty processes, q and r, such that for $u_p^i \leqslant t \leqslant \text{umax}^i$,*

$$C_q^i(t) - \alpha \leqslant C_p^{i+1}(t) \leqslant C_r^i(t) + \alpha,$$

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

Proof. $C_p^{i+1}(t) = C_p^i(t) + T^i + \delta - \text{AV}_p^i$. Therefore, by Lemma 6 there are nonfaulty processes q and r for which

$$C_p^i(t) + T^i + \delta - \text{ARR}_p^i(q) \leqslant C_p^{i+1}(t) \leqslant C_p^i(t) + T^i + \delta - \text{ARR}_p^i(r).$$

We show the right-hand inequality first. Let $a = c_p^i(\text{ARR}_p^i(r))$, the real time at which the message arrives at p from r. Thus, $C_p^i(a) = \text{ARR}_p^i(r)$. Note that $C_r^i(a) \geqslant T^i + (1-\rho)(\delta - \varepsilon)$.

$$\begin{aligned}
C_p^{i+1}(t) &\leqslant C_p^i(t) + T^i + \delta - \text{ARR}_p^i(r), \qquad \text{from above} \\
&= C_r^i(t) + C_p^i(a) - C_r^i(a) + T^i + \delta - \text{ARR}_p^i(r) \\
&\quad + (C_p^i(t) - C_r^i(t)) - (C_p^i(a) - C_r^i(a)) \\
&\leqslant C_r^i(t) + C_p^i(a) - C_r^i(a) + T^i + \delta - \text{ARR}_p^i(r) + 2\rho(t-a), \\
&\qquad \text{by Lemma 2 since } t > a \\
&\leqslant C_r^i(t) + \text{ARR}_p^i(r) - T^i - (1-\rho)(\delta - \varepsilon) \\
&\quad + T^i + \delta - \text{ARR}_p^i(r) + 2\rho(t-a) \\
&= C_r^i(t) + \varepsilon + \rho\delta - \rho\varepsilon + 2\rho(t-a).
\end{aligned}$$

It remains to bound $t-a$. The largest t can be is umax^i, and the smallest a can be is $\text{tmin}^i + \delta - \varepsilon$. So $t - a \leqslant \text{umax}^i - \text{tmin}^i - \delta + \varepsilon$. In the worst case, one process reaches T^i, then β later (by Theorem 4, part (c)), another process reaches T^i, and finally $(1+\rho)^2(\beta + \delta + \varepsilon)$ later, the second process reaches U^i. Thus, $\text{umax}^i - \text{tmin}^i \leqslant \beta + (1+\rho)^2(\beta + \delta + \varepsilon)$. Therefore, $t - a \leqslant \beta + (1+\rho)^2(\beta + \delta + \varepsilon) - \delta + \varepsilon$. Thus,

$$\begin{aligned}
C_p^{i+1}(t) &\leqslant C_r^i(t) + \varepsilon + \rho\delta - \rho\varepsilon + 2\rho(\beta + (1+\rho)^2(\beta + \delta + \varepsilon) - \delta + \varepsilon) \\
&= C_r^i(t) + \varepsilon + \rho(4\beta + \delta + 3\varepsilon) + 4\rho^2(\beta + \delta + \varepsilon) + 2\rho^3(\beta + \delta + \varepsilon) \\
&< C_r^i(t) + \alpha.
\end{aligned}$$

For the left-hand inequality, we see that $C_q^i(t) - \varepsilon - \rho\delta - \rho\varepsilon - 2\rho(t-a) \leqslant C_p^{i+1}(t)$, where $a = c_p^i(\mathrm{ARR}_p^i(q))$. The factor $t-a$ is bounded exactly as before, so that we obtain

$$C_q^i(t) - \alpha \leqslant C_p^{i+1}(t). \quad \blacksquare$$

The next lemma states a bound on the difference of processes' logical clocks with the same index, at a time when one of the processes is using that logical clock for its local time. This implies that the local times of two nonfaulty processes are close in those intervals where both use a clock with the same index.

LEMMA 15. *Let p and q be nonfaulty, $i \geqslant 0$. Then*

$$|C_p^i(t) - C_q^i(t)| \leqslant (1+\rho)(\beta + 2\rho(1+\rho)(\beta+\delta+\varepsilon))$$

for $\min\{u_p^{i-1}, u_q^{i-1}\} \leqslant t \leqslant \max\{u_p^i, u_q^i\}$, *if* $i \geqslant 1$, *and for* $\mathrm{tmin}^0 \leqslant t \leqslant \max\{u_p^0, u_q^0\}$, *if* $i = 0$.

Proof. Lemma 13 implies that

$$|c_p^i(T) - c_q^i(T)| \leqslant \beta + 2\rho(1+\rho)(\beta+\delta+\varepsilon)$$

for all T, $T^{i-1} - P - (1+\rho)(\beta+\varepsilon+\delta) \leqslant T \leqslant U^i$, if $i \geqslant 1$, and for all T, $T^0 - (1+\rho)(\beta+\delta+\varepsilon) \leqslant T \leqslant U^0$, if $i = 0$.

Case 1. $i = 0$. Lemma 3 implies the result for all t such that

$$\min\{c_p^0(T^0 - (1+\rho)(\beta+\delta+\varepsilon)), c_q^0(T^0 - (1+\rho)(\beta+\delta+\varepsilon))\}$$
$$\leqslant t \leqslant \max\{u_p^0, u_q^0\}.$$

It remains to show that

$$\mathrm{tmin}^0 \geqslant \min\{c_p^0(T^0 - (1+\rho)(\beta+\delta+\varepsilon)), c_q^0(T^0 - (1+\rho)(\beta+\delta+\varepsilon))\}.$$

By Lemma 1,

$$\begin{aligned} &c_p^0(T^0) - c_p^0(T^0 - (1+\rho)(\beta+\delta+\varepsilon)) \\ &\quad \geqslant \beta + \delta + \varepsilon \\ &\quad > \beta \\ &\quad \geqslant c_p^0(T^0) - \mathrm{tmin}^0, \qquad \text{by assumption A4 and definition of } \mathrm{tmin}^0. \end{aligned}$$

The result follows.

Case 2. $i \geqslant 1$. Let $S = T^{i-1} - P - (1+\rho)(\beta+\delta+\varepsilon)$. Lemma 3 implies the result for all t with $\min\{c_p^i(S), c_q^i(S)\} \leqslant t \leqslant \max\{u_p^i, u_q^i\}$. It remains to

show that $\min\{u_p^{i-1}, u_q^{i-1}\} \geq \min\{c_p^i(S), c_q^i(S)\}$. Now $u_p^{i-1} = c_p^{i-1}(U^{i-1}) = c_p^i(U^{i-1} + \mathrm{ADJ}_p^{i-1})$. By part (a) of Theorem 4 for i, $U^{i-1} + \mathrm{ADJ}_p^{i-1} \geq U^{i-1} - (1+\rho)(\beta+\varepsilon) - \rho\delta > T^{i-1} > S$. Since c_p^i is monotonic, $u_p^{i-1} > c_p^i(S)$. Similarly, one can show that $u_q^{i-1} > c_q^i(S)$, and the result follows. ∎

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

THEOREM 16. *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. We must show that $|L_p(t) - L_q(t)| \leq \gamma$, for all nonfaulty p and q, and all $t \geq \mathrm{tmin}^0$.

Case 1. $L_p(t) = C_p^i(t)$ and $L_q(t) = C_q^i(t)$, so p and q are using clocks with the same index. Thus t is such that

$$\max\{u_p^{i-1}, u_q^{i-1}\} \leq t < \min\{u_p^i, u_q^i\}, \text{ if } i \geq 1,$$

and

$$\mathrm{tmin}^0 \leq t < \min\{u_p^0, u_q^0\}, \text{ if } i = 0.$$

This case is covered by Lemma 15. 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.

Case 2. $L_p(t) = C_p^{i+1}$ and $L_q(t) = C_q^i(t)$ for some $i \geq 0$. This is the case where one of the processes has changed to a new clock, while the other still retains the old clock. We must bound $|C_p^{i+1}(t) - C_q^i(t)|$ for all t with $u_p^i \leq t \leq u_q^i$. Lemma 14 implies that there exist nonfaulty processes r and s such that

$$C_r^i(t) - \alpha \leq C_p^{i+1}(t) \leq C_s^i(t) + \alpha,$$

where

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

$$\begin{aligned} |C_p^{i+1}(t) - C_q^i(t)| &\leq \alpha + \max\{|C_r^i(t) - C_q^i(t)|, |C_s^i(t) - C_q^i(t)|\} \\ &\leq \alpha + (1+\rho)(\beta + 2\rho(1+\rho)(\beta + \delta + \varepsilon)), \qquad \text{by Lemma 15.} \end{aligned}$$

Lemma 15 is applicable since $u_p^i \leqslant t \leqslant u_q^i$ implies $u_q^{i-1} \leqslant t \leqslant u_q^i$, which implies

$$\min\{u_r^{i-1}, u_q^{i-1}\} \leqslant t \leqslant \max\{u_r^i, u_q^i\},$$

and similarly for s. The result follows. ▮

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 5.2. Assume P is fixed. The ith 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 Lemma 10. 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 $\beta/2 + 2\varepsilon + 2\rho P \leqslant \beta$ yields $\beta \geqslant 4\varepsilon + 4\rho P$.

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

If n increases while f remains fixed, a greater closeness of synchronization can be achieved by using the mean instead of the midpoint in the algorithm. Similarly to [DLPSW1], we can show that the convergence rate if the mean is used is roughly $f/(n-2f)$, and that an error of approximately 2ε is approachable.

8. Validity

The next major result is the validity condition, which states that clock time increases as a linear function of real time. Such a condition rules out trivial solutions to the clock synchronization problem such as periodically resetting all the clocks to 0.

The first lemma bounds the values of the zero-index clocks.

Lemma 17. $(1-\rho)(t - t_p^0) \leqslant C_p^0(t) - T^0 \leqslant (1+\rho)(t - t_p^0)$ *for* $t \geqslant t_p^0$.

Proof. By Lemma 1. ▮

The next lemma is the main one.

Lemma 18. *Let p be nonfaulty, $i \geqslant 0$. Then*

$$(1-\rho)(t - \text{tmax}^0) - i\varepsilon \leqslant C_p^i(t) - T^0 \leqslant (1+\rho)(t - \text{tmin}^0) + i\varepsilon$$

for all $t \geqslant u_p^{i-1}$ if $i \geqslant 1$, and for all $t \geqslant t_p^0$ if $i = 0$.

Proof. We proceed by induction on i. We argue the right-hand inequality. The left-hand inequality is entirely analogous.

Basis: $i=0$. This case follows immediately by Lemma 17, since $t_p^0 \geqslant \text{tmin}^0$.

Induction: Assume the result has been shown for $i \geqslant 0$ and show it for $i+1$.

Choose $t \geqslant u_p^i$. Note that $C_p^{i+1}(t) = C_p^i(t) + \text{ADJ}_p^i$. By Lemma 6 and the definition of ADJ_p^i, there exists a nonfaulty process r such that $\text{ADJ}_p^i \leqslant T^i + \delta - \text{ARR}_p^i(r)$. Let $a = c_p^i(\text{ARR}_p^i(r))$, the real time when p receives r's round i message. By the lower bound on the message delay, $a \geqslant t_r^i + \delta - \varepsilon$.

$$\begin{aligned} C_p^{i+1}(t) &\leqslant C_p^i(t) + T^i + \delta - \text{ARR}_p^i(r) \\ &= C_p^i(t) + T^i + \delta - C_p^i(a) \\ &\leqslant (1+\rho)(t-a) + T^i + \delta, \quad \text{by Lemma 1} \\ &\leqslant (1+\rho)(t - t_r^i - \delta + \varepsilon) + T^i + \delta, \quad \text{by lower bound on message delay} \\ &= (1+\rho)(t - \text{tmin}^0) - (1+\rho)(t_r^i - \text{tmin}^0) + T^i + \varepsilon - \rho\delta + \rho\varepsilon. \end{aligned}$$

By induction, $C_r^i(t_r^i) - T^0 \leqslant (1+\rho)(t_r^i - \text{tmin}^0) + i\varepsilon$, since $t_r^i \geqslant u_r^{i-1}$, if $i \geqslant 1$, and $t_r^i \geqslant t_r^0$ if $i=0$. Thus, $-(1+\rho)(t_r^i - \text{tmin}^0) \leqslant -C_r^i(t_r^i) + T^0 + i\varepsilon$. Thus,

$$\begin{aligned} C_p^{i+1}(t) &\leqslant (1+\rho)(t - \text{tmin}^0) - C_r^i(t_r^i) + T^0 + i\varepsilon + T^i + \varepsilon - \rho\delta + \rho\varepsilon \\ &\leqslant (1+\rho)(t - \text{tmin}^0) + T^0 + (i+1)\,\varepsilon, \text{ since } C_r^i(t_r^i) = T^i \text{ and } \delta \geqslant \varepsilon. \end{aligned}$$

The result follows. ∎

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

THEOREM 19. *The algorithm preserves* $(\alpha_1, \alpha_2, \alpha_3)$*-validity, where*

$$\alpha_1 = 1 - \rho - \varepsilon/\lambda, \qquad \alpha_2 = 1 + \rho + \varepsilon/\lambda, \qquad \alpha_3 = \varepsilon.$$

Proof. We must show for all $t \geqslant t_p^0$ and all nonfaulty p that

$$\alpha_1(t - \text{tmax}^0) - \alpha_3 \leqslant L_p(t) - T^0 \leqslant \alpha_2(t - \text{tmin}^0) + \alpha_3.$$

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

$$(1-\rho)(t - \text{tmax}^0) - i\varepsilon \leqslant C_p^i(t) - T^0 \leqslant (1+\rho)(t - \text{tmin}^0) + i\varepsilon.$$

Since $L_p(t)$ is equal to $C_p^i(t)$ for some i, we just need to convert i into an

expression in terms of t, etc. An upper bound on i is $1+(t-u_p^0)/\lambda$, since u_p^0 is when C_p^1 is first used. Since $u_p^0 > \text{tmax}^0$, $i < 1+(t-\text{tmax}^0)/\lambda$. Then

$$\begin{aligned}(1+\rho)(t-\text{tmin}^0)+i\varepsilon &\leqslant (1+\rho)(t-\text{tmin}^0)+(1+(t-\text{tmax}^0)/\lambda)\,\varepsilon \\ &\leqslant (1+\rho+\varepsilon/\lambda)(t-\text{tmin}^0)+\varepsilon, \qquad \text{since } \text{tmin}^0 \leqslant \text{tmax}^0,\end{aligned}$$

and

$$\begin{aligned}(1-\rho)(t-\text{tmax}^0)-i\varepsilon &\geqslant (1-\rho)(t-\text{tmax}^0)-(1+(t-\text{tmax}^0)/\lambda)\,\varepsilon \\ &\geqslant (1-\rho-\varepsilon/\lambda)(t-\text{tmax}^0)-\varepsilon.\end{aligned}$$

The result follows. ∎

9. Extensions

This section indicates how the basic algorithm presented above can be modified to allow a repaired process to synchronize its clock with the rest of the system, and how a variant of the algorithm can be used to establish synchronization initially. It also includes a brief discussion of an implementation of the basic algorithm.

9.1. *Reintegrating a Failed Process*

Our algorithm can be modified to allow a faulty process which has been repaired to synchronize its clock with the other nonfaulty processes. In this subsection, we give an overview; details may be found in [Lu1]. 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 as that 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. As soon as it awakens, it begins collecting T^i messages for all plausible values of T^i. It is necessary that p identify an appropriate round i at which it is able to obtain all the T^i messages from nonfaulty processes. Since p might awaken during the middle of a round, p will first orient itself by observing the arriving messages, allowing part of a round to pass before it begins to collect messages.

After p has determined that it should use T^i messages to update its clock, it continues to collect T^i messages. It must wait a certain length of time, as

measured on its clock, in order to guarantee that it has received T^i messages from all nonfaulty processes. 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.

9.2. *Establishing Synchronization*

In this subsection we present an algorithm to establish synchronization of clocks in a distributed system of processes, assuming the clocks initially have arbitrary values. The algorithm handles Byzantine failures of the processes, uncertainty in the message delivery time, and clock drift. The novel aspect of this approach is in *not* using the local times to trigger resynchronizations (as in the basic structure of our maintenance algorithm and those of [LM, HSSD]), but instead using a combination of elapsed physical time and number of special messages received.

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, rounds cannot be initiated when certain local times are reached, since processes' local times might be wildly far apart. Instead, each round contains an additional phase, in which the processes exchange messages to decide that they are ready to begin the next round. A more detailed description follows.

The algorithm guarantees that nonfaulty processes begin each round within real time $\delta + 3\varepsilon$ of each other. At the beginning of each round, each nonfaulty process p broadcasts its local time. Then p waits for an interval of length $(1+\rho)(2\delta + 4\varepsilon)$, which is long enough for the process 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, of length $(1+\rho)(4\varepsilon + 4\rho(\delta+2\varepsilon)+2\rho^2(\delta+4\varepsilon))$, 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. (The idea to use two criteria for ending the second interval came from [DLS].)

The code for the algorithm follows.

Local variables:

- A: initially arbitrary; adjustment to correction for current round.
- ASLEEP: initially true; boolean telling if process has been woken up yet or not.
- CORR: initially arbitrary (clocks are not synchronized); correction to physical clock.
- DIFF: initially arbitrary; array of estimated differences between this process' clock and the other processes' clocks, one entry per process.
- EARLY-END: initially arbitrary; boolean telling if second waiting interval was terminated early or not.
- RCVD-READY: initially empty; set of process ids of processes from whom this process has received a READY message during the current round.
- T: initially arbitrary; time at beginning of current round.
- U: initially -1; time in current round to cease collecting time messages and to compute the adjustment.
- V: initially -1; time in current round to broadcast READY.

Subroutines and global variables are the same as those for the previous algorithm. begin-round is a macro; its expansion is given first.

```
begin-round == /*macro*/
                  T := local-time( )
                  broadcast(T)
                  U := T + (1 + ρ)(2δ + 4ε)
                  set-timer(U)
                  EARLY-END := false
                  RCVD-READY := ∅
receive(START) and ASLEEP:
                  ASLEEP := false
                  begin-round
```

receive(T') from q:
    DIFF[q] := $T' + \delta -$ local-time( )
    if (ASLEEP) **then**
        ASLEEP := false
        begin-round
    endif
receive(TIMER) and (local-time( ) = U):
    A := mid(reduce(DIFF))
    $V := U + (1+\rho)(4\varepsilon + 4\rho(8+2\varepsilon) + 2\rho^2(\delta + 2\varepsilon))$
    set-timer(V)
receive(TIMER) and (local-time( ) = V) and (not EARLY-END):
    broadcast(READY)
receive(READY, q):
    RCVD-READY := RCVD-READY $\cup \{q\}$
    if ((|RCVD-READY| = $f+1$) and
      (local-time( ) < V)) **then**
        broadcast(READY)
        EARLY-END := true
    endif
    if (|RCVD-READY| = $n-f$) **then**
        DIFF := DIFF $- A$
        CORR := CORR $+ A$
        begin-round
    endif

A complete analysis of the closeness of synchronization attainable by this algorithm may be found in [Lu1]; here we merely state the main results.

Let B^i be the maximum difference between nonfaulty clock values at the latest real time when a nonfaulty process begins round i (i.e., when it broadcasts its clock value). As before, the fault-tolerant averaging function used in the algorithm causes the difference to be approximately halved at each round. More precisely (ignoring terms of order ρ^2),

LEMMA 20. *For* $i \geqslant 0$, $B^{i+1} \leqslant B^i/2 + 2\varepsilon + 2\rho(11\delta + 39\varepsilon)$.

Since the limit of B^i as the round number increases without bound is $4\varepsilon + 4\rho(11\delta + 39\varepsilon)$, we see that the algorithm achieves a closeness of synchronization of about 4ε.

Two modes of operation are possible. One is for the processes to run the start-up algorithm indefinitely. The other is to run the start-up algorithm just until the desired closeness of synchronization is achieved and then to switch the maintenance algorithm. A protocol to perform the switch between the algorithms may be found in [Lu1].

9.3. *Implementation*

A slightly modified version of the maintenance algorithm was implemented by the first author at AT & T Bell Laboratories during the summer of 1986. The program was written in the *C* language and was designed to synchronize the clocks of Suns running Berkeley Unix on an Ethernet. For a more complete description of the project, see [Lu2].

This project provided an interesting example of practice influencing theory—the algorithm had to be modified because of real-world constraints. Implementing the computational part of the algorithm was very easy. The challenging part was interacting with the operating system and the network, and trying to satisfy the assumptions of the model.

The major discrepancy between the model used in this paper and the reality of the Ethernet is the assumption of the existence of reliable, bounded delay communication, as well as a broadcast primitive. These appear to be mutually exclusive. Virtual circuits provide reliability, but the sending machine must connect to each recipient individually, and since the time to make each connection is nonzero, sending the same time in each message, as the algorithm requires, would not be correct. Another drawback of virtual circuits is that it is infeasible to keep n^2 virtual circuits open when n is big, due to limitations imposed by the size of certain data structures in the kernel.

On the other hand, broadcast is available using datagrams, but it is not reliable. Once datagrams arrive at a machine, they are stored in a buffer, and if too many arrive at once, the old ones are overwritten. In the algorithm of this paper, every process broadcasts as soon as its logical clock reaches a certain time. If the logical clocks are closely synchronized, all the processes will reach this value at about the same real time, and datagrams will get lost in the traffic jam, leading to the unfortunate situation that when the system behaves well, it is punished.

The solution to the problem is to use datagrams and to stagger the broadcast times. A spacing interval σ is chosen, and process p, for p between 0 and $n-1$, broadcasts at logical clock time $T^i + p\sigma$. The spacing interval should be big enough so that collisions are sufficiently infrequent that they can be attributed to faulty processes. Worst-case analysis shows that the modified algorithm behaves very similarly to the original one.

10. Comparison with Other Work

This section is a brief, and high-level, comparison of the maintenance algorithm presented in this paper with the interactive convergence algorithm of [LM] and the algorithms in [HSSD, M, MS, ST]. The

different assumptions made by the authors are pointed out, and various performance measures are discussed.

All the algorithms handle arbitrary (or "Byzantine") process faults, as long as $n > 3f$ (except where noted). They also all require known bounds on the message delays and clock drift, and that the processes be initially synchronized. For the rest of this section, we divide the algorihthms into two groups, those that need a fully connected network, and those that do not.

Our maintenance algorithm, and the algorithms in [LM, MS], assume a fully connected network. Since each process broadcasts at each round, n^2 messages are sent in every round. Our algorithm has already been described. Clocks stay synchronized to within about 4ε (ε is the uncertainty in the message delay, in our model). The synchronized clock's rate of drift does not exceed by very much the drift of the underlying hardware clocks. The size of the adjustment at each round is about 5ε. Rejoining is easy and a variant of the algorithm works when clocks are not initially synchronized.

The algorithm of Lamport and Melliar–Smith [LM] was described in the introduction. Let ε' be the uncertainty in the message delay in their model. The closeness of synchronization achieved is about $2n\varepsilon'$. Validity is close to that of the underlying hardware clocks (although it is not explicitly discussed). The size of the adjustment is about $(2n+1)\,\varepsilon'$. Superficially this performance looks worse than our algorithm's; however, in converting between the different models, it may be the case that $\varepsilon = n\varepsilon'$, because of the necessity of polling n queues for incoming messages [La2, LM]. This is an example of the many pitfalls encountered in comparing clock synchronization algorithms. Reintegration and initialization are not discussed in [LM].

The algorithms of Mahaney and Schneider [MS] are also based on the algorithm of [LM], and the model is the same. At each round, clock values are exchanged. All values that are not close enough to $n-f$ other values (thus are clearly faulty) are discarded, and the remaining values are averaged. However, the performance is analyzed in different terms, with more emphasis on how the clock values are related before and after a single round, so agreement, validity, and adjustment size values are not readily available for comparison. Reintegration and initialization are not discussed. A pleasing and novel aspect of this algorithm is that it degrades gracefully if more than one-third of the processes fail.

The next set of algorithms (those in [M, HSSD, ST]) does not require a fully connected network. Again, every process communicates with all its neighbors at each round, but since the network is not necessarily fully connected, the message complexity per round could be less than $O(n^2)$. The estimates of agreement, validity, and adjustment size presented in the rest

of this subsection for these algorithms are made assuming $n = 3f + 1$, and a fully connected network with no link failures, in order to facilitate comparison (although, as mentioned above, the algorithms do not require that these conditions hold).

The key idea of Marzullo's algorithm [M] is for each process to maintain an upper bound on the error of its clock. This bound allows an interval that includes the correct real time to be constructed. Periodically each process requests the time from each of its neighbors. As each response is received, the process sets its new interval to be the intersection of its current one with the interval received in response, after adjusting for further error that could be introduced by message delay. Since the algorithm's performance is analyzed probabilistically, assuming various probability distributions for the clock rates over time, it is difficult to compare results with our analysis, which makes worst-case assumptions.

The algorithm of Halpern, Simons, Strong, and Dolev [HSSD] can tolerate any number of process and link failures as long as the nonfaulty processes can still communicate. However, the price paid for this extra fault tolerance is that digital signatures are needed. When a process' clock reaches the next in a series of values (decided on in advance), the process begins the next round by broadcasting that value. If the process receives a message containing the value not too long before its clock reaches the value, it updates its clock to the value and relays the message. The closeness of synchronization achievable is about $\delta + \varepsilon$, which is either better or worse than our algorithm, depending on the relative sizes of δ and ε. The faulty processes, by sending messages too early, can cause the nonfaulty ones to speed up their clocks, and the slope of the synchronized clocks can exceed 1 by an amount that increases as f increases. The size of the adjustment is about $(f + 1)(\delta + \varepsilon)$, again depending on f. An algorithm to reintegrate a repaired process is mentioned; although it is complicated, it has the nice property of not forcing the process to wait possibly many hours until the next resynchronization , but instead starting as soon as the process requests it. No system initialization is discussed.

The algorithm of Srikanth and Toueg [ST] is very similar to that of [HSSD], but only handles less than $n/2$ process failures and does not handle link failures. However, they can relax the necessity of digital signatures (if $n > 3f$). Agreement, as in [HSSD] is about $\delta + \varepsilon$. Validity is optimal, i.e., is that provided by the underlying hardware clocks. The size of the adjustment is about $3(\delta + \varepsilon)$. There are twice as many messages per round as those in [HSSD] when digital signatures are not used. Reintegration is based on our method. A simple modification to the algorithm gives an elegant algorithm for initially synchronizing the clocks.

APPENDIX

This Appendix consists of definitions and lemmas concerning multisets needed for the proof of Lemma 9. These definitions and lemmas are analogous to some in [DLPSW1, DLPSW2].

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, $\operatorname{diam}(U)$, 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 $\operatorname{reduce}(U)$ 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, $\operatorname{mid}(U)$, is $\frac{1}{2}[\max(U) + \min(U)]$.

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 $\operatorname{mid}(U + r) = \operatorname{mid}(U) + r$ and $\operatorname{reduce}(U + r) = \operatorname{reduce}(U) + r$.

The first lemma bounds the diameter of a reduced multiset.

LEMMA 21. *Let U and W be multisets such that $|U| = n$, $|W| \geq n - f$, and $d_x(W, U) = 0$, where $n \geq 2f + 1$. Then*

$$\max(\operatorname{reduce}(U)) \leq \max(W) + x \quad \textit{and} \quad \min(\operatorname{reduce}(U)) \geq \min(W) - x.$$

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 is x-paired with an element 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 $\operatorname{reduce}(U)$ is x-paired with an element of W. Suppose $\max(\operatorname{reduce}(U))$ is x-paired with w in W by c. Then $\max(\operatorname{reduce}(U)) \leq w + x \leq \max(W) + x$.

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 $\max(\operatorname{reduce}(U)) \leq u \leq w + x \leq \max(W) + x$. ∎

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

LEMMA 22. *Let U and V be multisets, each with at least one element. Then*

$$d_x(l(U), l(V)) \leqslant d_x(U, V) \qquad \text{and} \qquad d_x(s(U), s(V)) \leqslant d_x(U, V).$$

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 $|S_x(c')| \leqslant |S_x(c)|$. Since $d_x(M, N) \leqslant |S_x(c')|$ and $|S_x(c)| = d_x(U, V)$, it follows that $d_x(M, N) \leqslant d_x(U, V)$.

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

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

Case 3. $c(u) \neq v$, 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 $|S_x(c')| \leqslant |S_x(c)|$.

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 $S_x(c)$, $|u - c(u)| = |u - v'| \leqslant x$. So $v' \geqslant u - x$. Hence $u' > v' + x \geqslant u - x + x$, which implies that $u' > u$. But this contradicts u being the largest element of U.

(ii) $v' > u' + x$. Since u' is not in $S_x(c)$, $|u' - c(u')| = |u' - v| \leqslant x$. So $u' \geqslant v - x$. Hence $v' > u' + x \geqslant v - x + x$, 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, cannot contain values that are too far apart.

LEMMA 23. *Let U, V, and W be multisets such that $|U| = |V| = n$ and $|W| \geqslant n - f$, where $n \geqslant 3f + 1$. If $d_x(W, U) = 0$ and $d_x(W, V) = 0$, then*

$$\min(\text{reduce}(U)) - \max(\text{reduce}(V)) \leqslant 2x.$$

Proof. First we show that $d_{2x}(U, V) \leqslant f$. 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 least $n-f$ elements w in W, there is u in U such that $u = c_U(w)$. Thus $|u - c(u)| \leqslant |u - w| + |w - c(u)| = |c_U(w) - w| + |w - c_V(w)| \leqslant x + x = 2x$. Thus $S_{2x}(c) \leqslant f$, so $d_{2x}(U, V) \leqslant f$.

Then by applying Lemma 22 f times, we know that $d_{2x}(\text{reduce}(U), \text{reduce}(V)) \leqslant f$. Since $|\text{reduce}(U)| = |\text{reduce}(V)| = n - 2f > f$, there are u in reduce(U) and v in reduce(V) such that $|u - v| \leqslant 2x$. Thus $\min(\text{reduce}(U)) - \max(\text{reduce}(V)) \leqslant u - v \leqslant 2x$. ∎

Lemma 24 is the main multiset result. It bounds the difference between the midpoints of two reduced multisets in terms of a particular third multiset.

LEMMA 24. *Let U, V, and W be multisets such that $|U| = |V| = n$ and $|W| \geqslant n - f$, where $n > 3f$. If $d_x(W, U) = 0$ and $d_x(W, V) = 0$, then*

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

Proof.

$$\begin{aligned}
&|\text{mid}(\text{reduce}(U)) - \text{mid}(\text{reduce}(V))| \\
&\quad = \tfrac{1}{2}\,|\max(\text{reduce}(U)) + \min(\text{reduce}(U)) \\
&\qquad - \max(\text{reduce}(V)) - \min(\text{reduce}(V))| \\
&\quad = \tfrac{1}{2}\,|\max(\text{reduce}(U)) - \min(\text{reduce}(V)) \\
&\qquad + \min(\text{reduce}(U)) - \max(\text{reduce}(V))|.
\end{aligned}$$

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

$$\begin{aligned}
&\tfrac{1}{2}[\max(\text{reduce}(U)) - \min(\text{reduce}(V)) + \min(\text{reduce}(U)) - \max(\text{reduce}(V))] \\
&\quad \leqslant \tfrac{1}{2}(\max(W) + x - (\min(W) - x) + \min(\text{reduce}(U)) \\
&\qquad - \max(\text{reduce}(V))), \qquad \text{by applying Lemma 21 twice} \\
&\quad = \tfrac{1}{2}(\text{diam}(W) + 2x + \min(\text{reduce}(U)) - \max(\text{reduce}(V))) \\
&\quad \leqslant \tfrac{1}{2}(\text{diam}(W) + 2x + 2x), \qquad \text{by Lemma 23} \\
&\quad = \tfrac{1}{2}\text{diam}(W) + 2x.
\end{aligned}$$

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

Nomenclature

ADJ	variable holding the current adjustment to the current correction of the physical clock. (4.2)
ADJ_p^i	value of ADJ calculated by p at round i, and used to create the $(i+1)$st logical clock, $i \geqslant 0$. (5.2)
$ARR[1 \cdots n]$	array holding arrival times at a process of most recent messages, one entry for each process. (4.2)
$ARR_p^i(q)$	arrival time, measured on p's logical clock C_p^i, of T^i message from q, $i \geqslant 0$. (5.2)
ARR_p^i	multiset of values $ARR_p^i(q)$ for all q, $i \geqslant 0$. (5.2)
AV	variable equal to mid(reduce(ARR)), "average" arrival time of T^i messages, for current i. (4.2)
AV_p^i	value of AV calculated by process p at round i, $i \geqslant 0$. (5.2)
C, D	clocks.
C_p^0	process p's initial (or 0th) logical clock, Ph_p + initial value of CORR. (3.2)
C_p^i	process p's ith logical clock, equal to $C_p^{i-1} + ADJ_p^{i-1}$ for $i > 0$. (5.2)
clock	monotonically increasing, everywhere differentiable function from $\mathbb{R}$ to $\mathbb{R}$. (2.1)
CORR	variable holding the current software correction to the physical clock. (3.2 and 4.2)
$CORR_p(t)$	value of process p's CORR variable at real time t in some execution. (3.2)
d_x	the minimum, over all correspondences between two multisets, of the number of elements u of one multiset that are mapped to elements v of the other such that $\lvert u - v \rvert > x$. (Appendix)
diam	difference between the largest and the smallest elements of a multiset. (Appendix)
f	upper bound on number of faulty processes. (3.2)
$L_p(t)$	process p's local time at real time t, $Ph_p(t) + CORR_p(t)$. (3.2)
local time	synchronized time for a process, physical clock time plus current correction. (3.2)
logical clock	the physical clock plus a correction value. (3.2)
mid	midpoint of the interval spanned by a multiset. (4.2 and Appendix)
n	total number of processes, $n > 3f$. (3.2)
P	length of a round in local time. (4.1)
p, q, r	processes.

$\mathrm{Ph}_p(t)$	physical clock of process p. (2.1)
physical clock	process' hardware clock, not under its control. (2.1)
reduce	multiset resulting after removing the f largest and f smallest elements in a multiset. (4.2 and Appendix)
round	for a given process, the interval between its ith logical clock reaching T^i and its $(i+1)$st logical clock reaching T^{i+1} is its round i, $i \geq 0$. (4.1)
START message	"message" sent by the environment to wake up a process initially. (2.1)
t, t_1, t_2	real times
T, T_1, T_2	clock times.
T^i	clock time at which each process begins round i and broadcasts the value T^i; $T^i = T^0 + iP$, $i \geq 0$. (3.2 and 5.2)
t_p^i	real time at which process p begins round i, $i \geq 0$. (5.2)
tmax^i	max over all nonfaulty p of t_p^i, latest real time when a nonfaulty process begins round i, $i \geq 0$. (3.2 and 5.2)
tmin^i	min over all nonfaulty p of t_p^i, earliest real time when a nonfaulty process begins round i, $i \geq 0$. (3.2 and 5.2)
TIMER message	"message" indicating an interrupt received by the process because a certain amount of physical clock time elapsed. (2.1)
U^i	clock time $T^i + (1+\rho)(\beta + \delta + \varepsilon)$, $i \geq 0$, at which round i update to clock is done, producing C_p^{i+1}. (5.2)
u_p^i	real time at which process p updates its clock in round i, $i \geq 0$. (5.2)
umax^i	max over nonfaulty p of u_p^i, latest real time when a nonfaulty process updates its clock in round i, $i \geq 0$. (5.2)
umin^i	min over nonfaulty p of u_p^i, earliest real time when a nonfaulty process updates its clock in round i, $i \geq 0$. (5.2)
α	variable used locally in several proofs to simplify expressions.
$\alpha_1, \alpha_2, \alpha_3$	parameters in validity condition. (3.2)
β	maximum difference in real time between nonfaulty processes beginning the algorithm. (3.2)
γ	agreement parameter, upper bound on closeness of local times. (3.2)
δ	midpoint of range of possible message delays. (3.2)
ε	uncertainty in message delay (all delays are between $\delta - \varepsilon$ and $\delta + \varepsilon$). (3.2)
λ	lower bound on length of a round in real time. (8)
ρ	rate of drift of the physical clock, in clock seconds per real seconds. (3.1)

ACKNOWLEDGMENTS

We thank Gene Stark and Bill Weihl for their comments on an earlier version of part of this paper, and Mark Tuttle for a careful proofreading. Alan Fekete contributed many ideas for simplifying the proofs and presentation. The referees' comments helped improve the presentation as well. A preliminary version of this paper appears in the "Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing, August 27–29, 1984, Vancouver, B. C.," pp. 75–88. (c) ACM. Any portions of this paper that appeared in the original version are reprinted with permission. This work was supported in part by the NSF under Grant DCR-8302391, U.S. Army Research Office Contracts DAAG-79-C-0155 and DAAG29-84-K-0058, Advanced Research Projects Agency of the Department of Defense Contract N00014-83-K-0125, and Office of Naval Research Contract N00014-85-K-0168.

RECEIVED July 25, 1985; ACCEPTED May 19, 1987

REFERENCES

[DHS] DOLEV, D., HALPERN, J., AND STRONG, R. (April 1986), On the possibility and impossibility of achieving clock synchronization, *J. Comput. System Sci.* **22** (2), 230–250.

[DLPSW1] DOLEV, D., LYNCH, N., PINTER, S., STARK, E., AND WEIHL, W. (1983), Reaching approximate agreement in the presence of faults, *in* "Proceedings of the Third Annual IEEE Symposium on Distributed Software and Database Systems."

[DLPSW2] DOLEV, D., LYNCH, N., PINTER, S., STARK, E., AND WEIHL, W. (July 1986), Reaching approximate agreement in the presence of faults, *J. Assoc. Comput. Mach.* **33** (3), 499–416.

[DLS] DWORK, C., LYNCH, N., AND STOCKMEYER, L. (1984), Consensus in the presence of partial synchrony, *in* "Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing," pp. 103–118.

[HSSD] HALPERN, J., SIMONS, B., STRONG, R., AND DOLEV, D. (1984), Fault-tolerant clock synchronization, *in* "Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing," pp. 89–102.

[La1] LAMPORT, L. (July 1978), Time, clocks, and the ordering of events in a distributed system, *Comm. ACM* **21** (7), 558–565.

[La2] LAMPORT, L., personal communication.

[LM] LAMPORT, L., AND MELLIAR-SMITH, P. M. (January 1985), Synchronizing clocks in the presence of faults, *J. Assoc. Comput. Mach.* **32** (1), 52–78.

[Lu1] LUNDELIUS, J. (1984), "Synchronizing Clocks in a Distributed System," S. M. thesis, MIT, MIT/LCS/TR-335.

[Lu2] LUNDELIUS, J. (1986), Software clock synchronization in a distributed system, manuscript.

[MS] MAHANEY, S., AND SCHNEIDER, F. (1985), Inexact agreement: Accuracy, precision, and graceful degradation, *in* "Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing," pp. 237–249.

[M] MARZULLO, K. (1983), "Loosely-Coupled Distributed Services: A Distributed Time Service," Ph. D. dissertation, Stanford University.

[ST] SRIKANTH, T. K., AND TOUEG, S. (1985), Optimal clock synchronization, *in* "Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing," pp. 71–86.