MIT/LCS/TM-265

A NEW FAULT-TOLERANT ALGORITHM FOR CLOCK SYNCHRONIZATION

Jennifer Lundelius

Nancy Lynch

July 1984

A New Fault-Tolerant Algorithm for Clock Synchronization*

Jennifer Lundelius

Nancy A. Lynch

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

June 1984

ABSTRACT

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 a 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 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.

*This work was supported in part by the NSF under Grant No. DCR-8302391, U.S. Army Research Office Contracts #DAAG29-79-C-0155 and #DAAG29-84-K-0058, and Advanced Research Projects Agency of the Department of Defense Contract #N00014-83-K-0125.

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. Taking into account the clocks' drift from real time and varying message delivery times makes the problem more realistic and more challenging. In order to be truly useful, a solution to this problem must allow faulty processes that have recovered to be reintegrated into the system. The algorithm described in this paper 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 processes are totally connected for communication. They communicate by messages, over a reliable transmission medium. There are upper and lower bounds on the length of time that any message takes to arrive at its destination. We do not require the existence of unforgeable signatures.

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 [DLPSW] 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 [LM], 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.

Lamport and Melliar-Smith [LM], Halpern, Simons and Strong [HSS], and Marzullo [M] also have clock synchronization algorithms that run in rounds. The three algorithms in [LM], as do ours, require a reliable, completely connected communication network and handle arbitrary faults. However, the closeness of synchronization achieved by one depends on the number of processes and that achieved by the other two depends on the number of faulty processes. In two of them, the size of the adjustment also depends on the number of faulty processes and the number of messages is exponential. Although one algorithm only needs a majority of the processes to be nonfaulty, it assumes unforgeable digital signatures. The algorithm of [HSS] is resilient to any number of faults (as long as the network remains connected), has n^2 message complexity per round, and achieves a closeness of synchronization very similar to ours. But the size of the adjustment depends on the number of processes and unforgeable digital signatures are necessary. The framework and error model used in [M] make a direct comparison of results with ours difficult. Only [HSS] includes a reintegration procedure.

The problem addressed in the earlier papers 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. A variant of the algorithm in this paper can be used to establish the initial synchronization, as well as to maintain the synchronization. This variant, together with a description of the interface between the two algorithms, will be briefly sketched.

The remainder of this paper is organized as follows: in Section 2 we describe the underlying model upon which our work is based in more detail, but still informally. 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 described in Section 2. The algorithm to solve the problem is presented in Section 4. This simple algorithm is described in words first, and then in a high level "programming language". We explain how the high level language can be "compiled" into our model. Section 5 contains an inductive proof that some important properties hold at every round. We give an upper bound on the amount by which any nonfaulty process' clock is changed at any time. Section 6 includes background needed for the results of Section 7, which contains the answers to the problem posed earlier. In section 8 we explain how to reintegrate a repaired process. Finally, Section 9 consists of a brief description of an algorithm to establish synchronization initially.

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 will appear in [Lu].

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.

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 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 define a *clock* to be a monotonically increasing, everywhere differentiable function from $\mathbb{R}$ (real time) to $\mathbb{R}$ (clock time). A *system of processes* consists of a set of processes, a subset of the processes called the *self-starting processes*, and a set of clocks (the physical clocks), one 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. Initially the message buffer contains no messages except for START messages, exactly one for each self-starting process.

When a process p sets a timer, say for time T, a TIMER message with recipient p and delivery time $Ph_p^{-1}(T)$, is placed in the message buffer, as long as $Ph_p^{-1}(T)$ is not less than the current real time. If it is, 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 the configurations of the system immediately before the event and immediately afterwards, e.g. (F,e,F'), is an *action*.

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

- the configurations match up correctly, that is, the second configuration of an action is the same as the first one of the following action;
- 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");
- if an action (F, 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 messages 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 time $Ph_p(t)$;
- if any process p sets a timer for a future time t, then at time t, p receives a TIMER message; furthermore, if any nonfaulty process p receives a TIMER message at time t, then earlier p set a timer for t; and
- a message m is received at real time t if and only if the message buffer contained m with t recorded as the time at which it was to be delivered.

Since faulty processes need not obey the conditions in the third and fourth properties listed above, they can choose when they take steps and can do anything they want at a step.

3. The Clock Synchronization Problem

3.1. Clocks

In this paper, clock names are capitalized. For each clock, the inverse function has the same name, but it is not capitalized.

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

$$1-\rho \leq 1/(1+\rho) \leq C'(t) \leq 1+\rho \leq 1/(1-\rho).$$

Henceforth we assume that all clocks are ρ-bounded, i.e., the amount by which a clock's rate is faster or slower than real time is at most ρ.

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

Lemma 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 2: Let C and D be clocks.

(a) If $C' = 1$ 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 $C' = 1$ 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 1. ▮

Lemma 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 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

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

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

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

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

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 $CORR_p(t)$, 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. In keeping with our notational convention, we let c^0_p denote the inverse function of C^0_p. 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 are part of a logical clock.

3.2. Problem Statement

We make the following assumptions:

(1) 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.)

(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 $3f + 1$. (Dolev, Halpern and Strong [DHS] show that it is impossible without authentication to synchronize clocks unless more than 2/3 of the processes are nonfaulty.)

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

(4) 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. The Algorithm

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 [DLPSW] 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.

4.2. 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.

```
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 + $(1 + \rho)(\beta + \delta + \varepsilon)$)

     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 + $\delta$ - AV
     CORR := CORR + ADJ
     set-timer(T + P)
     endstep
     beginstep(u)
     enddo
```

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. The set-timer statement takes an argument U which represents a logical time. The corresponding physical time, U − CORR, is the physical time which is described by the transition function.

5. Inductive Analysis

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

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.

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 $\min_{p \text{ nonfaulty}}\{t^i_p\}$, 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.

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: 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 1. ∎

Lemma 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 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 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 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. ∎

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 7: Let p be nonfaulty. Then $U^i + ADJ^i_p < T^{i+1}$.

Proof: $U^i + ADJ^i_p \le U^i + (1+\rho)(\beta+\varepsilon) + \rho\delta$, by Lemma 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.

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

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

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

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

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

$= \varepsilon$.

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 $\beta/2$, in a rough sense. 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 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 23. 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

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

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 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) = β.

By Lemma 23, $|\text{mid}(\text{reduce}(U)) - \text{mid}(\text{reduce}(V))| \leq \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon)$.

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

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

Lemma 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

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

$$\leq \beta/2 + 2\varepsilon + 2\rho(\beta + \delta + \varepsilon) + 2\rho(1 + \rho)(\beta + \delta + \varepsilon)\text{, by Lemmas 9 and 2.}$$

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 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 2

$\leq \rho((1 + \rho)(\beta + \varepsilon) + \rho\delta)$, by Lemma 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 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 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 2 and 11.

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

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 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.

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 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 2 and assumption 4

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

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 2 and 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 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 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 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:

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

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.

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 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 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 3 immediately implies the needed result for $i = 0$.

Induction: $i \geq 1$. Lemma 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 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 1

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

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

To bound the second term we note that Lemma 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 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 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 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. ∎

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 5.1. 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 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 [DLPSW], 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.

7.2. Validity

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

Lemma 18: $T^0 + (1-\rho)(t - t^0_p) \leq C^0_p(t) \leq T^0 + (1+\rho)(t - t^0_p)$ for $t \geq t^0_p$.

Proof: By Lemma 1. ∎

The next lemma is the main one.

Lemma 19: 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 18.

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 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 20: 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. ∎

8. 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. 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. 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 p 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. More specifically, p first 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.

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 continues to collect 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 slight extra bookkeeping is necessary because T^i messages from nonfaulty processes can arrive at p before p has received the f-th T^{i-1} message. We omit a description of a scenario in which this occurs.)

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.

(It is also necessary to argue that when p resets its clock, the new clock has not already reached T^{i+1}. We assume that P is big enough to ensure this. We haven't shown that the lower bound on P given earlier is sufficient.)

9. Establishing Synchronization

In this section we present an algorithm to synchronize 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. We envision the processes running this algorithm until the desired degree of synchronization is obtained, and then switching to the maintenance algorithm.

9.1. 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. A more detailed description follows.

Nonfaulty processes will 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 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 [DLS].)

It is apparent that 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.

Let B^i be the maximum difference between nonfaulty clock values at the latest real time when a nonfaulty process begins round i. Ignoring terms of order ρ^2, we can bound B^{i+1} in terms of B^i as follows:

$$B^{i+1} \leq \tfrac{1}{2}B^i + 2\varepsilon + 2\rho(13\delta + 43\varepsilon).$$

The idea of the proof is similar to the proof of Theorem 17. Again, the fault-tolerant averaging function used in the algorithm causes the difference to be approximately halved at each round.

By considering the limit of B^i as the round number increases without bound, we can show that the algorithm achieves a closeness of synchronization of about $4\varepsilon + 4\rho(13\delta + 43\varepsilon)$.

As for the maintenance algorithm, if we use the mean instead of the midpoint in this algorithm, we can approach an error of about 2ε as n increases and f remains fixed.

9.2. 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.

Now each process does Byzantine Agreement on the vector of NROUNDS values, one 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 justification is as follows: 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.

9.3. Switching to the Maintenance Algorithm

After the processes have done the required number of rounds, say r, of this algorithm to establish synchronization, they must begin the maintenance algorithm. Remember that that algorithm works by having each process broadcast 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. The processes should begin the maintenance algorithm as soon as possible in order to minimize the inaccuracy introducted by the clock drift.

It can be shown 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 the first multiple of P is reached, each process broadcasts its clock value as in the maintenance algorithm, but doesn't update its clock. At the second multiple of P, each process begins the full maintenance algorithm by broadcasting its clock value and updating its clock. (It will receive clock values from all nonfaulty processes.) There will be a lag of at most one round between any two nonfaulty processes' beginning the maintenance algorithm. Then β, the difference in real time between two nonfaulty processes reaching T^i, can be calculated from B^r, the fact that all processes begin the algorithm at most 2P in clock time after $tmax^r$, and the result of Lemma 15 that clocks that are reset one round early don't change by too much. This β will be slightly larger than the smallest one maintainable. To shrink it back down, P can be made slightly smaller than required by the maintenance algorithm.

Mike Fischer has suggested using only the algorithm to establish synchronization and not using the maintenance algorithm at all. Further work is needed to investigate this idea; however, it may be reasonable since both algorithms synchronize to approximately 4ε.

Acknowledgements

Thanks to Gene Stark and Bill Weihl for their comments on an earlier version of part of this paper.

References

[DHS] D. Dolev, J. Halpern and R. Strong, On the possibility and impossibility of achieving clock synchronization, Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing (1984).

[DLPSW] D. Dolev, N. Lynch, S. Pinter, E. Stark and W. Weihl, Reaching approximate agreement in the presence of faults, Proceedings of the Third Annual IEEE Symposium on Distributed Software and Database Systems (1983).

[DLS] C. Dwork, N. Lynch and L. Stockmeyer, Consensus in the presence of partial synchrony, to appear in Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (1984).

[HSS] J. Halpern, B. Simons and R. Strong, Fault-tolerant clock synchronization, to appear in Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (1984).

[L] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM, Vol. 21, No. 7 (July 1978).

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

[Lu] J. Lundelius, Synchronizing clocks in a distributed system, S.M. thesis, MIT (in progress).

[M] K. Marzullo, Loosely-coupled distributed services: a distributed time service, Ph.D. dissertation, Stanford University (1983).

Appendix

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

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 21: Let U and W be multisets such that $|U| = |W| = n$ and $d_x(U,W) \leq f$, 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 $d_x(U,W)$. Suppose none of the f elements deleted from the high end of U are x-paired with elements of W by c. Since $d_x(W,U) \leq f$, 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)) ≤ u ≤ w + x ≤ max(W) + x. ∎

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 22: 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)) ≤ 2x.

Proof: First we show that there is a w in W such that w is x-paired both with some u in reduce(U) and with some v in reduce(V) by the mappings witnessing $d_x(W,U)$ and $d_x(W,V)$ respectively. We know $|reduce(U)| = |reduce(V)| = n - 2f$ and $|W| = n - f$. In order to choose two disjoint subsets of size n – 2f from a set of size n – f, it must be the case that $n - f \geq 2(n - 2f)$. But this implies that $n \leq 3f$, contradicting the hypothesis.

By choice of u, v, and w, we know that $|u - w| \leq x$ and $|v - w| \leq x$. Thus, min(reduce(U)) − max(reduce(V)) $\leq u - v \leq w + x - (w - x) = 2x$. ▮

Lemma 23 is the main multiset result. It bounds the difference between the midpoints of two reduced multisets in terms of a particular third multiset.

Lemma 23: 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 |mid(reduce(U)) − mid(reduce(V))| $\leq$ ½diam(W) + 2x.

Proof: |mid(reduce(U)) − mid(reduce(V))|

= ½|max(reduce(U)) + min(reduce(U)) − max(reduce(V)) − min(reduce(V))|

= ½|max(reduce(U)) − min(reduce(V)) + min(reduce(U)) − max(reduce(V))|

If the quantity inside the absolute value signs is nonnegative,

= ½[max(reduce(U)) − min(reduce(V)) + min(reduce(U)) − max(reduce(V))|

$\leq$ ½[max(W) + x − (min(W) − x) + min(reduce(U)) − max(reduce(V))], by applying Lemma 21 twice

= ½[diam(W) + 2x + min(reduce(U)) − max(reduce(V))]

$\leq$ ½[diam(W) + 2x + 2x], by Lemma 22

= ½diam(W) + 2x.

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