messages aren't received until after the time when p sets its (i + 1)-st clock.

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

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

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

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

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

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

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

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

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

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

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

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

and that

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

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

The result follows. ∎

4.8 Reintegrating a Repaired Process

Our algorithm can be modified to allow a faulty process which has been repaired to synchronize its clock with the other nonfaulty processes. Let p be the process to be reintegrated into the system. During some round i, p will gather messages from the other processes and perform the same averaging procedure described previously to obtain a value for its correction variable such

that its clock becomes synchronized. Since p's clock is now synchronized, it will reach T^{i+1} within β of every other nonfaulty process. At that point, p is no longer faulty and rejoins the main algorithm, sending out T^{i+1} messages.

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

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

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

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

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

message.

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

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

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

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

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

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

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

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

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

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

The code is in Figure 4-2.

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

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

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

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

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

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

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

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

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

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

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

There are two ways to solve this problem:

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

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

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

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

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

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

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

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

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

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

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

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

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

This inequality simplifies to the stated bound. ∎

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

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

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

$$P \geq (5\beta + \delta + 10\varepsilon + 2\rho(5\beta + 2\delta + 9\varepsilon)) / (2 \cdot 4\rho), \text{ ignoring } \rho^2 \text{ terms.}$$

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

Chapter Five

Establishing Synchronization

5.1 Introduction

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

5.2 The Algorithm

5.2.1 General Description

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

Nonfaulty processes will begin each round within real time $\delta + 3\varepsilon$ of each other. Each nonfaulty process begins the algorithm, and its round 0, as soon as it first receives a message. (It will be shown that this must be within $\delta + 3\varepsilon$.) 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