Optimal Clock Synchronization†

T.K. Srikanth
Sam Toueg

TR 84-656
December 1984
(Revised May 1985)

Department of Computer Science
Cornell University
Ithaca, New York 14853

† Partial support for this work was provided by the National Science Foundation under grant MCS 83-03135.

To appear in: Journal of the ACM

Optimal Clock Synchronization†

T.K. Srikanth
Sam Toueg

Department of Computer Science
Cornell University
Ithaca, New York 14853

ABSTRACT

We present a simple, efficient and unified solution to the problems of synchronizing, initializing, and integrating clocks, for systems with different types of failures: crash, omission, and arbitrary failures with and without message authentication. This is the first known solution that achieves optimal accuracy, i.e., the accuracy of synchronized clocks (with respect to real time) is as good as that specified for the underlying hardware clocks. The solution is also optimal with respect to the number of faulty processes that can be tolerated to achieve this accuracy.

1. Introduction

An important problem in distributed computing is that of synchronizing clocks in spite of faults. Given "hardware" clocks whose rate of drift from real time is within known bounds, synchronization consists of maintaining logical clocks that are never too far apart. Processes maintain these logical clocks by computing periodic adjustments to their hardware clocks.

Although the underlying hardware clocks have a bounded rate of drift from real time, the drift of logical clocks can exceed this bound. In other words, synchronized logical clocks can have a lower accuracy (with respect to real time) than that specified for hardware clocks. This reduction in accuracy might appear to be an inherent consequence of synchronization. The rate of drift of faulty hardware clocks can be beyond the specified bounds, and correct logical clocks can be forced to drift with them. Furthermore, variation in message delivery times introduces uncertainty in evaluating values of clocks of other processes. All previous synchronization algorithms exhibit this reduction in accuracy [Lamp85, Halp84, Lund84, Dole84].

In this paper we show that accuracy need not be sacrificed in order to achieve synchronization. We present the first synchronization algorithm where logical clocks have the same accuracy as the underlying physical clocks. We show that no synchronization algorithm can achieve a better accuracy, and therefore our algorithm is optimal in this respect.

---

† Partial support for this work was provided by the National Science Foundation under grant MCS 83-03135.

In contrast to previous results, we present a unified solution to the different versions of the problem: systems that exhibit crash, omission, or arbitrary failures with and without message authentication. With simple modifications, the solution also provides for initial clock synchronization and for the integration of new clocks.

We first present an algorithm for systems with arbitrary failures assuming that the system provides authentication. We then develop a broadcast primitive that achieves those properties of authentication required by the algorithm [Srik84]. Replacing authenticated communication with this primitive results in an equivalent non-authenticated algorithm. This solution is then simplified for crash and omission failures.

We show that to achieve optimal accuracy, fewer than half the clocks in the system can be faulty. With arbitrary failures, and in the absence of authentication, synchronization can be achieved only if fewer than a third of the clocks in the system are faulty [Dole84]. Our algorithm is optimal with respect to the number of faulty clocks it can tolerate for all the models of failure that we consider.

The solution presented in this paper is simple and efficient, and its message complexity is comparable to those previously published. Further comparisons with previous results are presented in Section 7.

The paper is organized as follows. We describe the system model in Section 2. In Section 3, we describe an authenticated synchronization algorithm that achieves optimal accuracy, and we derive bounds on the number of faults that can be tolerated to achieve this accuracy. In Section 4, we present a broadcast primitive that simulates authenticated broadcasts, and we use it to get a non-authenticated synchronization algorithm. Initialization and integration are discussed in Section 5. Crash and omission models of failure are considered in Section 6. Discussion of the results and concluding remarks are presented in Sections 7 and 8.

2. The model

We consider a system of distributed processes that communicate through a reliable, error-free and fully connected message system (the connectivity condition is relaxed later). Each process has a physical "hardware" clock and computes its logical time by adding a locally determined adjustment to this physical clock.

The notation used here closely follows that in [Halp84]. Variables and constants associated with *real* time are in lower case and those corresponding to the *logical* time of a process are in upper case. The following assumptions are made about the system:

A1. The rate of drift of physical clocks from real time is bounded by a known constant $\rho > 0$. That is, if $R_i(t)$ is the reading of the physical clock of process i at time t, then for all $t_2 \geq t_1$,

$$(1+\rho)^{-1}(t_2 - t_1) \leq R_i(t_2) - R_i(t_1) \leq (1+\rho)(t_2 - t_1)$$

Thus, correct physical clocks are within a linear envelope of real time. We also see that the rate of drift between clocks is bounded by $dr = \rho(2+\rho)/(1+\rho)$.

A2. There is a known upper bound t_{del} on the time required for a message to be prepared by a process, sent to all processes and processed by the correct processes receiving it.

A process is faulty if it deviates from its algorithm or if its physical clock violates assumption A1, otherwise it is said to be correct. Faulty processes may also collude to prevent correct processes from achieving synchronization. We use the term "correct clock" to refer to the logical clock of a correct process.

Resynchronization proceeds in rounds, a period of time in which processes exchange messages and reset their clocks. A process i starts a new logical clock C_i^k after the k^{th} resynchronization. Define beg^k and end^k to be the real time at which the first and last correct process respectively start their k^{th} clocks. The period $[beg^k, end^k]$ is the k^{th} *resynchronization period.*

Given the above assumptions, a *synchronization algorithm* is one that satisfies the following conditions for all correct clocks i and j, all $k \geq 1$, and $t \in [end^k, end^{k+1}]$:

1. *Agreement*: There exists a constant $D_{\max}$ such that

$$\left| C_i^k(t) - C_j^k(t) \right| \leq D_{\max}$$

2. *Accuracy*: There exists a constant γ such that for any execution of the algorithm,

$$(1+\gamma)^{-1}t + a \leq C_i^k(t) \leq (1+\gamma)t + b$$

for some constants a and b which depend on the initial conditions of this execution.

The *agreement* condition asserts that the maximum deviation between correct logical clocks is bounded. The *accuracy* condition states that correct logical clocks are within a linear envelope of real time.

Note that γ is a bound on the rate of drift of logical clocks from real time and hence is a measure of their accuracy with respect to real time. We are interested in synchronization algorithms that minimize γ. In Theorem 2, we show that γ cannot be smaller than ρ, the bound on the accuracy of physical clocks. Therefore, we are interested in algorithms satisfying:

3. *Optimal accuracy*: For any execution of the algorithm, for all correct clocks i, all $k \geq 1$, and $t \in [end^k, end^{k+1}]$

$$(1 + \rho)^{-1} t + a \leq C_i^k(t) \leq (1 + \rho) t + b$$

for some constants a and b which depend on the initial conditions of this execution.

3. The authenticated algorithm

The following is an informal description of a synchronization algorithm for systems with n processes of which at most f are faulty. The algorithm requires that $n \geq 2f+1$ and that messages are authenticated. Informally, authentication prevents a faulty process from changing a message it relays, or introducing a new message into the system and claiming to have received it from some other process.

Let P be the logical time between resynchronizations. A process expects the k^{th} resynchronization, for $k \geq 1$, at time kP on its logical clock. When $C^{k-1}(t) = kP$, it broadcasts a signed message of the form (*round* k), indicating that it is ready to resynchronize. When a process receives such a message from $f+1$ distinct processes, it knows that at least one correct process is ready to resynchronize. It is then said to *accept* the message, and decides to resynchronize, even if its logical clock has not yet reached kP. A process resynchronizes by starting its k^{th} clock, setting it to $kP + \alpha$, where α is a constant. To ensure that clocks are never set back, α is chosen to be greater than the increase in C^{k-1} since the process sent a (*round* k) message. After resynchronizing, the process also relays the $f+1$ signed (*round* k) messages to all other processes to ensure that they also resynchronize. The algorithm is described in Figure 1. We show that it achieves *agreement* and *accuracy*. We later modify it to achieve *optimal accuracy*.

```
cobegin
  if C^{k-1}(t) = kP                    /* ready to start C^k */
      → sign and broadcast (round k) fi
//
  if accepted the message (round k)     /* received f+1 signed (round k) messages */
      → C^k(t) := kP + α;               /* start C^k */
        relay all f+1 signed messages to all fi
coend
```

Figure 1. An authenticated algorithm for clock synchronization for process p for round k.

3.1. Proof of correctness: Agreement

We first show that the algorithm achieves the *agreement* property. Define $ready^k$ to be the earliest (real) time at which any correct process sends a (*round k*) message. We assume that the clocks C^0 of correct processes are synchronized, i.e., at $ready^1$ all correct processes are using clock C^0 and for all correct processes i and j, $\left|C_i^0(ready^1) - C_j^0(ready^1)\right| \le D_{\max}$. In Section 5, we describe an algorithm for achieving this initial synchronization. For ease of presentation we assume that the maximum permitted deviation between correct logical clocks, $D_{\max}$, is a given constraint.

Lemma 1: The k^{th} resynchronization period is bounded in size. That is, there exists a constant $d_{\min}$ such that for $k \ge 1$, $end^k - beg^k \le d_{\min}$.

Proof: Let p be the first correct process to start its k^{th} clock. By definition, this occurs at beg^k. Process p must have received $f+1$ signed (*round k*) messages. Since it relays all these messages, every correct process receives them and accepts the message (*round k*) by time $beg^k + t_{del}$. Hence, every correct process starts its k^{th} clock by time $beg^k + t_{del}$. By setting $d_{\min} = t_{del}$, we get $end^k \le beg^k + d_{\min}$. □

Lemma 2: At the end of the k^{th} resynchronization period, correct clocks differ by at most $d_{\min}(1+\rho)$. That is, for $k \ge 1$, and for all correct processes i and j, $\left|C_i^k(end^k) - C_j^k(end^k)\right| \le d_{\min}(1+\rho)$.

Proof: By Lemma 1, $end^k - beg^k \le d_{\min}$. Therefore the last correct process to start its k^{th} clock does so within $d_{\min}$ of the first correct clock doing so, and in this period, the first clock could have drifted by at most $\rho d_{\min}$. Thus, at end^k, the difference between correct clocks is at most $d_{\min}(1+\rho)$. □

Lemma 3: No correct process starts its k^{th} clock until at least one correct process is ready to do so, i.e., $beg^k \ge ready^k$, for $k \ge 1$.

Proof: The first correct process to start its k^{th} clock does so only when it accepts a (*round k*) message, i.e., only when it receives (*round k*) messages from at least $f+1$ processes. Since at least one correct process must have sent a (*round k*) message, $beg^k \ge ready^k$. □

Assume that the following conditions hold for some $k \ge 1$:

S1. At $ready^k$ all correct processes are using C^{k-1}.

S2. For correct processes i and j, $\left|C_i^{k-1}(ready^k) - C_j^{k-1}(ready^k)\right| \le D_{\max}$.

With these assumptions, we prove the following lemmas.

Lemma 4: All correct processes start their k^{th} clocks soon after one correct process is ready to do so. Specifically, $end^k - ready^k \le (1+\rho)D_{\max} + t_{del}$.

Proof: The first correct process to send a (*round k*) message does so at $ready^k$. By S2, the slowest correct clock is no more than $D_{\max}$ behind. Hence, every correct process sends a (*round k*) message no later than $(1 + \rho)D_{\max}$ after $ready^k$, and therefore every correct process starts its k^{th} clock within a further t_{del}. Thus, $end^k - ready^k \le (1 + \rho)D_{\max} + t_{del}$. □

By Lemma 4, the real time that elapses from the time a correct process sends a (*round k*) message (when C^{k-1} reads kP) to the time it starts C^k (setting it to $kP + \alpha$) is at most $(1 + \rho)D_{\max} + t_{del}$. Therefore, if $\alpha \ge \left[(1 + \rho)D_{\max} + t_{del}\right](1 + \rho)$ then no correct process sets its logical clock backwards. Henceforth, we assume that α satisfies this relation.

Lemma 5: There is a bound on the period for which the k^{th} logical clock is used. That is, $end^{k+1} - end^k \le (P - \alpha)(1 + \rho) + t_{del}$.

Proof: Every correct process that sends a (*round* $k+1$) message does so no later than the time $(k+1)P$ on its clock, i.e., no later than $(P - \alpha)(1 + \rho)$ after end^k. Every process starts its $k+1^{st}$ clock within a further t_{del}, thus proving the lemma. □

Lemma 6: The maximum deviation between the k^{th} logical clocks of correct processes i and j is bounded. That is, for $t \in [end^k, end^{k+1}]$, $\left|C_i^k(t) - C_j^k(t)\right| \le D_{\max}$.

Proof: By Lemma 2, correct logical clocks are at most $d_{\min}(1 + \rho)$ apart at end^k. By Lemma 5, $end^{k+1} - end^k \le (P - \alpha)(1 + \rho) + t_{del}$, and clocks of correct processes can drift apart at a rate dr in this interval. Thus, in the interval $[end^k, end^{k+1}]$,

$$\left|C_i^k(t) - C_j^k(t)\right|$$
$$\le [(P - \alpha)(1 + \rho) + t_{del}]dr + d_{\min}(1 + \rho)$$
$$\le [P(1 + \rho) + t_{del}]dr + d_{\min}(1 + \rho)$$

P is chosen to satisfy the relation $D_{\max} \ge [P(1 + \rho) + t_{del}]dr + d_{\min}(1 + \rho)$. □

Lemma 7: Synchronization periods do not overlap. That is, $end^k < ready^{k+1} \le beg^{k+1}$.

Proof: The first correct process to send a (*round* $k+1$) message does so no earlier than at real time $beg^k + (P - \alpha)/(1 + \rho)$. Therefore, $ready^{k+1} \ge beg^k + (P - \alpha)/(1 + \rho)$. Hence, by Lemma 1, $ready^{k+1} \ge end^k - d_{\min} + (P - \alpha)/(1 + \rho)$. By Lemma 3, $ready^{k+1} \le beg^{k+1}$. Thus, $end^k < ready^{k+1} \le beg^{k+1}$, if P satisfies the relation $P > d_{\min}(1 + \rho) + \alpha$. □

From the proof of Lemmas 6 and 7, we see that $D_{\max}$ cannot be made arbitrarily small. The proof of Lemma 6 shows that $D_{\max} \ge [(P - \alpha)(1 + \rho) + t_{del}]dr + d_{\min}(1 + \rho)$. From Lemma 7, we see that $P - \alpha \ge d_{\min}(1 + \rho)$. Therefore, the smallest possible $D_{\max}$ that this algorithm can achieve is given by $D_{\max} \ge d_{\min}(1 + \rho)^3 + t_{del}dr$. It has been shown that, for any algorithm, $D_{\max}$ must be at least $t_{del}/2$ [Dole84].

Lemma 8: The algorithm in Figure 1 achieves *agreement*.

Proof: If assumptions S1 and S2 hold for some $k \geq 1$, then Lemma 6 states that the *agreement* condition is satisfied for k. We now show, by induction on k, that S1 and S2 hold for all $k \geq 1$ and therefore *agreement* is satisfied for all $k \geq 1$. As stated earlier, our initialization algorithm will guarantee that S1 and S2 are true for the base case, $k=1$.

Assume that S1 and S2 are true for some k. By Lemma 7, $end^k < ready^{k+1} \leq beg^{k+1}$. Thus, at $ready^{k+1}$, all correct processes use their k^{th} clocks. From Lemma 6 it follows that at $t=ready^{k+1}$, and for correct processes i and j, $\left|C_i^k(t)-C_j^k(t)\right| \leq D_{\max}$. Thus S1 and S2 are true for $k+1$. □

3.2. Proof of correctness: Accuracy

We now show that the algorithm achieves *accuracy*.

Lemma 9: For any execution of the algorithm of Figure 1, there exists a constant b, such that for all correct processes i, all $k \geq 1$ and for $t \in [end^k, end^{k+1}]$:

$$C_i^k(t) \leq \frac{P}{P-\alpha}(1 + \rho)t+b$$

Proof: Let $E(t_0)$ be the set of executions of the algorithm in which $ready^1=t_0$. Consider an execution $e \in E(t_0)$ in which for all $k \geq 1$, $ready^k=beg^k$, and the clock of correct process j, C_j^k, is started at beg^k. In execution e, the physical clock of process j runs at the maximum possible rate, i.e., $(1 + \rho)$ with respect to real time. It is clear that execution e is possible.

Since C_j^k is started at beg^k for each k, it is started at least as early as any other correct C_i^k in execution e. Furthermore, between beg^k and beg^{k+1}, C_j^k increases at the maximum possible rate. Hence, C_j^k is an upper bound on the k^{th} logical clocks of all correct processes in execution e. That is, for $t \in [end^k, end^{k+1}]$, $C_i^k(t) \leq C_j^k(t)$, for any other correct process i.

We now show that C_j^k is an upper bound on the k^{th} logical clock of any correct process in any execution in $E(t_0)$. To prove this, we first show that for any $k \geq 1$, $ready^k$ in execution e is at least as early as $ready^k$ in any other execution $e' \in E(t_0)$. The proof is by induction on k.

For $k=1$, $ready^1=t_0$ for all executions in $E(t_0)$. Assume, for some $k>1$, that $ready^k$ in execution e is no later than $ready^k$ in execution e'. In execution e, $beg^k=ready^k$, the k^{th} logical clock of process j is started at beg^k, and process j runs at the maximum possible rate. Therefore, $ready^{k+1}=ready^k+(P-\alpha)/(1 + \rho)$. It is easy to show that in any execution, $ready^{k+1} \geq ready^k+(P-\alpha)/(1 + \rho)$. Therefore, $ready^{k+1}$ in execution e is at least as early as that in execution e'.

In execution e, $beg^k=ready^k$ for all $k \geq 1$. By Lemma 3, in any execution, $beg^k \geq ready^k$ for all $k \geq 1$. Therefore, the k^{th} logical clock of process j is started no later than that of any other correct process in any execution in $E(t_0)$. Since process j also runs at the maximum possible rate, C_j^k is an upper bound on the k^{th} logical clocks of all correct processes in all executions in $E(t_0)$.

We now estimate an upper bound for C_j^k. For process j, the interval of real time between consecutive resynchronizations is $(P-\alpha)/(1+\rho)$. In this period its logical time increases by P. Therefore, for all $k \geq 1$, and for $t \in [end^k, end^{k+1}]$:

$$\frac{C_j^k(t) - C_j^1(t_0)}{t - t_0} \leq \frac{P}{P-\alpha}(1+\rho)$$

Since $C_j^1(t_0) = P + \alpha$, a constant, $C_j^k(t) \leq \frac{P}{P-\alpha}(1+\rho)t + b$, where b is a constant that depends on t_0.

□

Lemma 10: For any execution of the algorithm of Figure 1, there exists a constant a, such that for all correct processes i, all $k \geq 1$ and for $t \in [end^k, end^{k+1}]$:

$$\frac{P}{P - \alpha + \frac{t_{del}}{(1+\rho)}}(1+\rho)^{-1}t + a \leq C_i^k(t)$$

Proof: Let $E(t_0)$ be the set of executions of the algorithm in which $end^1 = t_0$. Consider an execution $e \in E(t_0)$ where, for all $k \geq 1$, correct process j accepts the *(round k)* message t_{del} in real time after C_j^{k-1} reads kP. Also, C_j^k is started at end^k for all $k \geq 1$. In e, the physical clock of process j runs at the minimum possible rate, i.e., at $(1+\rho)^{-1}$ with respect to real time. Such an execution is clearly possible. It is easy to show that C_j^k is a lower bound on the k^{th} logical clocks of all correct processes in execution e. That is, for $t \in [end^k, end^{k+1}]$, $C_j^k(t) \leq C_i^k(t)$, for any other correct process i.

C_j^k is also a lower bound on the k^{th} logical clocks of all correct processes in any execution in $E(t_0)$. In execution e we have $end^{k+1} = end^k + (P-\alpha)(1+\rho) + t_{del}$. The proof follows by Lemma 5 and an easy induction on k.

We now estimate a lower bound for C_j^k. For process j, $(P-\alpha)(1+\rho) + t_{del}$ is the interval of real time between consecutive resynchronizations. In this period, its logical time increases by P. Therefore, as in Lemma 9, for all $k \geq 1$ and $t \in [end^k, end^{k+1}]$

$$C_j^k(t) \geq \frac{P}{(P-\alpha)(1+\rho) + t_{del}}t + a$$

for some constant a which depends on t_0. □

Theorem 1: The algorithm in Figure 1 is a synchronization algorithm. With this algorithm, correct processes send a total of $O(n^2 f)$ signed messages per resynchronization.

Proof: By Lemma 8, the algorithm achieves *agreement*. Lemmas 9 and 10 imply that *accuracy* is achieved with $(1+\gamma)=\frac{P}{P-\alpha}(1+\rho)$. In each resynchronization round, each correct process broadcasts at most one signed message and relays at most $f+1$ signed messages to every other process. Thus, correct processes send a total of $O(n^2f)$ signed messages per resynchronization. □

The number of bits exchanged for each resynchronization is comparable to that in [Halp84].

3.3. Achieving optimal accuracy

3.3.1. A bound on accuracy

We first show that for any synchronization algorithm, the accuracy of synchronized logical clocks cannot exceed that of the underlying hardware clocks. In what follows, define $C_i(t)=C_i^k(t)$ for $t\in[end^k, end^{k+1})$ and all $k \geq 1$.

Theorem 2: For any synchronization algorithm, the bound on the rate of drift of logical clocks from real time is at least as large as the bound on the rate of drift of physical clocks.

Proof: Consider an algorithm that satisfies *agreement* and *accuracy*. For simplicity, assume that all physical clocks are set to 0 at time $t=0$, i.e., $R_i(0)=0$ for all i. Then, all correct physical clocks satisfy the relation

$$(1+\rho)^{-1}t \leq R_i(t) \leq (1+\rho)t$$

Consider an execution of the algorithm in which all processes in the system are correct and the physical clock of each process runs at the maximum possible rate. That is, for all processes i, $R_i^{(1)}(t)=(1+\rho)t$, where superscripts denote execution numbers. Further, assume the transmission delay for each message is exactly d, with $d \leq t_{del}/(1+\rho)^2$. By *accuracy*, in this execution, for all correct processes i and for some constant $b^{(1)}$:

$$C_i^{(1)}(t) \leq (1+\gamma)t+b^{(1)} \tag{1}$$

Now consider a second execution in which all processes are still correct, but have their physical clocks running at the minimum possible rate. That is, for all processes j, $R_i^{(2)}(t)=(1+\rho)^{-1}t$. Let the transmission delay for each message be $d(1+\rho)^2$. Again, by *accuracy*, in this execution for all correct processes i and for some constant $a^{(2)}$:

$$(1+\gamma)^{-1}t+a^{(2)} \leq C_i^{(2)}(t) \tag{2}$$

Assume that for each process i, the initial state is the same in both executions. That is, in both executions, a process starts executing the algorithm at the same reading of its physical clock. In the second execution, physical clocks and the speed at which messages are delivered are slowed down by the same factor, $(1+\rho)^2$, with respect to the first execution.

Therefore, from within the system both executions appear identical to every process. Hence, considering a particular process i, the rate at which its logical time advances with respect to its physical time must be the same in both executions. In particular, if $R_i^{(1)}(t_1)=R_i^{(2)}(t_2)$ for some t_1 and t_2, then $C_i^{(1)}(t_1)=C_i^{(2)}(t_2)$.

Since $R_i^{(1)}(t)=(1 + \rho)t$ and $R_i^{(2)}(t)=(1 + \rho)^{-1}t$, it follows that if $t_2=(1 + \rho)^2 t_1$, then $R_i^{(1)}(t_1)=R_i^{(2)}(t_2)$ and therefore $C_i^{(1)}(t_1)=C_i^{(2)}(t_2)$. Therefore, from equations (1) and (2), $(1 + \gamma)t_1+b^{(1)} \geq (1 + \gamma)^{-1}(1 + \rho)^2 t_1+a^{(2)}$ for all t_1. This implies that $\gamma \geq \rho$. □

3.3.2. An algorithm for optimal accuracy

We now describe a modification to our algorithm to achieve *optimal accuracy*. In the algorithm of Figure 1, correct processes start their k^{th} clocks as soon as they accept a (*round k*) message. However, there is an uncertainty of t_{del} in the time it takes for correct processes to accept a message. It is this uncertainty that introduces a difference in the logical time between resynchronizations. For the fastest clock, the logical time between resynchronizations is $P-\alpha$ (Lemma 9), and for the slowest clock, this interval is $P-\alpha+\dfrac{t_{del}}{(1 + \rho)}$ (Lemma 10). Informally, we can compensate for this as follows: if a process accepts a (*round k*) message early, it delays the starting of the k^{th} clock by $\dfrac{t_{del}}{2(1 + \rho)}$. If it accepts the message late, it advances the starting of the k^{th} clock by $\dfrac{t_{del}}{2(1 + \rho)}$. Thus, in the cases described in both Lemmas 9 and 10, the logical time between resynchronizations becomes $P-\alpha+\dfrac{t_{del}}{2(1 + \rho)}$, as will be shown later. This will then be used to achieve *optimal accuracy*.

More precisely, suppose process i accepts (*round k*) at time t, and let $T=C_i^{k-1}(t)$. Define $\beta=\dfrac{t_{del}}{2(1 + \rho)}$.

If $T \leq kP+\beta$, we say the (*round k*) message was accepted *early*. Process i delays the starting of C_i^k by setting it to $kP+\alpha$ when C_i^{k-1} reads $\min(T+\beta,\ kP+\beta)$. In this case, the start of C_i^k is delayed by at most β but never beyond the time when C_i^{k-1} reads $kP+\beta$.

If $T>kP+\beta$, we say (*round k*) was accepted *late*. Process i advances the starting of C_i^k, by setting it to $kP+\alpha$ when C_i^{k-1} reads $\max(T-\beta,\ kP+\beta)$. Note that C_i^k must be started when C_i^{k-1} reads $T'<T$, that is, "in the past". This is achieved by setting C_i^k to $kP+\alpha+(T-T')$ when C_i^{k-1} reads T. That is, C_i^k is set to $\min(C_i^{k-1}(t)+\alpha-\beta, kP+\alpha+\beta)$ at time t. In this case, the start of C_i^k is advanced by at most β, but is never started before C_i^{k-1} reads $kP+\beta$.

The definitions of $ready^k$, beg^k and end^k are the same as before: $ready^k$ is the earliest time at which a correct process sends a (*round* k) message; beg^k and end^k are the earliest and latest times at which some correct process starts its k^{th} clock (setting it to $kP+\alpha$).

We first show that this modified algorithm achieves *agreement* by showing that Lemmas 1 to 8 still hold.

Proof of Lemma 1: The first correct process to start its k^{th} clock can start it β in logical time (or $\beta(1 + \rho)$ in real time) before it accepts a (*round* k) message. Every correct process accepts (*round* k) within t_{del} of the first correct process accepting it, and starts its k^{th} clock within a further $\beta(1 + \rho)$. Therefore, $end^k - beg^k \le t_{del} + 2\beta(1 + \rho) = 2t_{del}$. Therefore, Lemma 1 is satisfied with $d_{\min} = 2t_{del}$.

Proof of Lemma 2: As in Section 3.

Proof of Lemma 3: Consider any correct process i. By definition, $C_i^{k-1}(ready^k) \le kP$. Let process i accept the (*round* k) message at real time t. Note that $t \ge ready^k$. If $C_i^{k-1}(t) \le kP+\beta$, then process i delays the starting of the k^{th} clock. If $C_i^{k-1}(t) > kP+\beta$, process i starts its k^{th} clock no earlier than at real time t' such that $C_i^{k-1}(t') = kP+\beta$. Clearly, $t' \ge ready^k$. Hence, no correct process starts its k^{th} clock before $ready^k$.

Proof of Lemma 4: Every correct process that broadcasts a (*round* k) message does so by real time $t_1 = ready^k + (1 + \rho)D_{\max}$. Therefore, every correct process accepts (*round* k) by $t_2 = t_1 + t_{del}$. For any correct process i, $C_i^{k-1}(t_1) \ge kP$ and hence $C_i^{k-1}(t_2) \ge kP + t_{del}/(1 + \rho) = kP + 2\beta$. Thus, with the modified algorithm, every correct process starts its k^{th} clock at real time $t < t_2$. Therefore, $end^k - ready^k \le (1 + \rho)D_{\max} + t_{del}$.

Proof of Lemma 5: Consider any correct process i. Process i accepts a (*round* $k+1$) message by real time $t = end^k + (P-\alpha)(1 + \rho) + t_{del}$. Also, $C_i^k(t) \ge kP + t_{del}/(1 + \rho)$. Therefore, process i starts its $k+1^{st}$ clock by real time t, proving the lemma.

Proofs of Lemmas 6, 7 and 8: As in Section 3.

Thus, the modified algorithm achieves *agreement*. To show that the modified algorithm achieves *optimal accuracy*, we first evaluate the bounds on the drift of logical clocks from real time.

Lemma 9': For any execution of the modified algorithm, there exists a constant d, such that for all correct processes i, all $k \ge 1$ and $t \in [end^k, end^{k+1}]$:

$$C_i^k(t) \le \frac{P}{P-\alpha+\beta}(1 + \rho)t + d$$

Proof: Let $E(t_0)$ be the set of executions of the algorithm in which $ready^1 = t_0$. Consider an execution $e \in E(t_0)$ in which for all $k \ge 1$, correct process j broadcasts and accepts (*round* k) at $ready^k$. In execution e, the physical clock of process j runs at the maximum possible rate, i.e., $(1 + \rho)$ with respect to real time.

Process j accepts $(round\ k)$ at $ready^k$, when C_j^{k-1} reads kP (i.e., early). Therefore, C_j^k is started at real time t such that $C_j^{k-1}(t) = kP + \beta$, i.e., when $t = ready^k + \beta/(1 + \rho)$. Note that no correct physical clock increases by more than β between $ready^k$ and t.

Consider another correct process i. By definition of $ready^k$, $C_i^{k-1}(ready^k) \leq kP$, and therefore, $C_i^{k-1}(t) \leq kP + \beta$. Suppose process i accepts $(round\ k)$ when C_i^{k-1} reads T_i. This must occur after $ready^k$, and therefore, at time t, $C_i^{k-1}(t) \leq T_i + \beta$. We consider two cases:

1. If $T_i \leq kP + \beta$, then process i starts C_i^k at real time t' when $C_i^{k-1}(t') = \min(T_i + \beta, kP + \beta)$. Since both $C_i^{k-1}(t) \leq T_i + \beta$ and $C_i^{k-1}(t) \leq kP + \beta$, then $t \leq t'$.

2. If $T_i > kP + \beta$, process i starts C_i^k at real time t' when $C_i^{k-1}(t') = \max(T_i - \beta, kP + \beta)$. Therefore, $C_i^{k-1}(t') \geq kP + \beta \geq C_i^{k-1}(t)$ and $t' \geq t$.

Thus, in execution e, for any $k \geq 1$, the k^{th} clock of process i is started no earlier than that of process j. Between resynchronizations, C_j^k runs at the maximum possible rate. Therefore, C_j^k is an upper bound on the k^{th} logical clock of all correct processes in execution e. As in Lemma 9, we can also show that C_j^k is an upper bound on the k^{th} clock of all correct processes in any execution in $E(t_0)$.

Between every two successive resynchronizations, the logical clock of process j is advanced by P, and the time that elapses on the logical clock of j is $P - \alpha + \beta$. (For example, at the k^{th} resynchronization, the clock is set to $kP + \alpha$, the $k+1^{st}$ resynchronization occurs when this clock reads $(k+1)P + \beta$, and the new clock is set to $(k+1)P + \alpha$.) Since the clock of process j runs at $(1 + \rho)$ with respect to real time, the real time that elapses between two resynchronizations is $(P - \alpha + \beta)/(1 + \rho)$. Hence, for all $k \geq 1$ and $t \in [end^k, end^{k+1}]$:

$$C_j^k(t) \leq \frac{P}{P - \alpha + \beta}(1 + \rho)t + d$$

for some constant d that depends on t_0. □

Lemma 10': For any execution of the modified algorithm, there exists a constant c, such that for all correct processes i, all $k \geq 1$ and for $t \in [end^k, end^{k+1}]$:

$$\frac{P}{P - \alpha + \beta}(1 + \rho)^{-1}t + c \leq C_i^k(t)$$

Proof: Let $E(t_0)$ be the set of executions of the algorithm in which $end^1 = t_0$. Define $last^k$ to be the latest real time at which a correct process accepts $(round\ k)$. Consider an execution $e \in E(t_0)$ in which the first logical clock of a correct process j, C_j^1 is started at end^1, and for all $k \geq 1$, process j accepts $(round\ k)$ at $last^k$, and t_{del} (in real time) after its logical clock reads kP. The physical clock of process j runs at the minimum possible rate, i.e., at $(1 + \rho)^{-1}$ with respect to real time. In the modified algorithm, since $C_j^{k-1}(last^k) = kP + 2\beta$, process j sets its

k^{th} clock to $kP+\alpha+\beta$ at $last^k$.

We now show that the logical clock of process j is as slow as that of any other correct process in any execution in $E(t_0)$. That is, we show that for all $k \geq 1$ and $t \in [last^k, last^{k+1}]$, $C_j^k(t) \leq C_i^k(t)$ for any correct process i in any execution in $E(t_0)$. The proof is by induction on k.

For $k=1$, note that C_j^1 is started at $end^1 = t_0$ and process j runs at the minimum possible rate. In any execution in $E(t_0)$, for any other correct process i, C_i^1 is started no later than at end^1. Therefore, for $t \geq end^1$, and specifically for $t \in [last^1, last^2]$, we see that $C_j^1(t) \leq C_i^1(t)$. For the inductive step, assume that for some $k>1$ and $t \in [last^{k-1}, last^k]$, we have $C_j^{k-1}(t) \leq C_i^{k-1}(t)$ for any correct process i. Define s_i to be the real time such that $C_i^{k-1}(s_i) = kP+\beta$, for any process i. Let t_i and T_i be the real and the corresponding logical time at which a process i accepts (*round* k).

Consider any correct process i in any execution in $E(t_0)$. From the induction hypothesis, it follows that $s_i \leq s_j$ for all correct i. Since $s_j = last^k - \beta(1+\rho)$, $last^k - s_i \geq \beta(1+\rho)$. By assumption, $t_j = last^k$ and $T_j = kP+2\beta$. We consider two cases:

1. If $T_i \leq kP+\beta$ (i.e., $t_i \leq s_i \leq s_j$), then C_i^k is set to $kP+\alpha$ no later than s_i. Since $last^k - s_i \geq \beta(1+\rho)$, C_i^k increases by at least β between s_i and $last^k$. Therefore, $C_i^k(last^k) \geq kP+\alpha+\beta = C_j^k(last^k)$.

2. If $T_i > kP+\beta$, then process i sets its k^{th} clock to $C_i^k(t_i) = \min(C_i^{k-1}(t_i)+\alpha-\beta, kP+\alpha+\beta)$. Since C_i^k and C_i^{k-1} increase by the same amount between t_i and $last^k$, $C_i^k(last^k) = \min(C_i^{k-1}(t_i)+\alpha-\beta, kP+\alpha+\beta) + C_i^{k-1}(last^k) - C_i^{k-1}(t_i)$. Since $C_i^{k-1}(last^k) \geq kP+2\beta$, $C_i^k(last^k) \geq kP+\alpha+\beta = C_j^k(last^k)$.

Thus, $C_j^k(last^k) \leq C_i^k(last^k)$. The physical clock of process j runs at the minimum possible rate. Therefore, for $t \in [last^k, last^{k+1}]$, $C_j^k(t) \leq C_i^k(t)$ for any correct process i in any execution in $E(t_0)$.

The logical clock of process j is incremented by P over successive resynchronizations. The real time that elapses between successive resynchronizations of process j is $(P-\alpha+\beta)(1+\rho)$. Thus, for any execution of the modified algorithm, there exists a constant c (that depends on t_0), such that for all correct processes i, all $k \geq 1$ and $t \in [last^k, last^{k+1}]$,

$$\frac{P}{P-\alpha+\beta}(1+\rho)^{-1}t + c \leq C_j^k(t)$$

Since for $t \in [end^k, last^k]$ $C_j^k(t) \geq C_j^{k-1}(t)$, the above inequality also holds for $t \in [end^k, end^{k+1}]$.

□

By Lemmas 9' and 10', in any execution of the algorithm, for $k \geq 1$ and for $t \in [end^k, end^{k+1}]$, the logical clock of any correct process i is within the envelope

$$\mu(1+\rho)^{-1}t+c \le C_i^k(t) \le \mu(1+\rho)t+d$$

where $\mu = \dfrac{P}{P-\alpha+\beta}$, c and d are constants depending on the initial conditions of this execution. Therefore,

$$(1+\rho)^{-1}t+c/\mu \le C_i(t)/\mu \le (1+\rho)t+d/\mu$$

Hence, if correct processes slow down their logical clocks by this factor of μ, i.e., process i uses $L_i(t)=C_i(t)/\mu$ as its logical time, *optimal accuracy* is achieved. Also, since $\mu>1$, *agreement* is still guaranteed. Process i continues to use C_i for the synchronization algorithm.

Theorem 3: With the modification described above, the algorithm of Figure 1 achieves *optimal accuracy.*

Proof: Follows from the above discussion. □

3.4. Bounds on faults tolerated

We now consider the maximum number of faults that can be overcome by a synchronization algorithm that achieves *optimal accuracy.*

Theorem 4: Any synchronization algorithm that achieves *optimal accuracy* must have a majority of correct clocks.

Proof: Assume that there exists a synchronization algorithm that achieves *optimal accuracy* for systems with $n \le 2f$. We show that this is impossible by first considering a system with two processors p_1 and p_2, one of which can be faulty (i.e., $n=2$ and $f=1$).

Since the algorithm achieves *optimal accuracy*, in any execution of the algorithm, the logical clock of correct process i satisfies the following relation for all $t \ge end^1$:

$$(1+\rho)^{-1}t+a \le C_i(t) \le (1+\rho)t+b$$

where a and b are constants. Also, since the algorithm achieves *agreement*, there exists a constant $D_{\max}$ such that if p_1 and p_2 are correct, then $|C_1(t)-C_2(t)| \le D_{\max}$ for all $t \ge end^1$.

We now consider three possible executions of the algorithm. In what follows, superscripts correspond to execution numbers. For simplicity, we assume that all physical clocks start at 0 at real time 0. Assume that the initial state of a given process is the same in all executions. That is, a given process starts executing the algorithm at the same reading of its physical clock.

Execution e_1: Both processes are correct. The physical clock of p_1 runs at the maximum rate possible and that of p_2 at the minimum rate possible. That is, $R_1^{(1)}(t)=(1+\rho)t$ and $R_2^{(1)}(t)=(1+\rho)^{-1}t$. The transmission time for each message is exactly d, where $d \le t_{del}/(1+\rho)^2$.

Execution e_2: Process p_1 is correct and the rate of its physical clock is given by $R_1^{(2)}(t)=(1+\rho)^{-1}t$. The clock of p_2 is faulty and runs at $R_2^{(2)}(t)=(1+\rho)^{-3}t$, but p_2 is otherwise correct and follows the algorithm. The transmission time of each message is $d(1+\rho)^2$.

Execution e_3: Process p_2 is correct and its physical clock is given by $R_2^{(3)}(t)=(1+\rho)t$. The clock of p_1 is faulty and runs at $R_1^{(3)}(t)=(1+\rho)^3t$, but p_1 is otherwise correct. All messages now take $d/(1+\rho)^2$ to be delivered.

We see that all three executions are possible. Since *optimal accuracy* is achieved, and since p_1 is correct in e_1, its logical clock satisfies the relation $C_1^{(1)}(t) \le (1+\rho)t+b^{(1)}$. Since $R_1^{(1)}(t)=(1+\rho)t$, we see that $C_1^{(1)}(t) \le R_1^{(1)}(t)+b^{(1)}$. Similarly, in execution e_2, we see that $R_1^{(2)}(t)+a^{(2)} \le C_1^{(2)}(t)$. But the two executions look identical to p_1, and hence the relation between its logical and physical clocks must be the same in both executions. Therefore, to satisfy the two relations above, we see that for $k=1,2$,

$$R_1^{(k)}(t)+a^{(2)} \le C_1^{(k)}(t) \le R_1^{(k)}(t)+b^{(1)}$$

Therefore, in execution e_1, there exists a time τ such that for all $t \ge \tau$

$$(1+\rho)t+a^{(2)} \le C_1^{(1)}(t) \le (1+\rho)t+b^{(1)}$$

Similarly, by considering executions e_1 and e_3, in both of which p_2 is correct, we see that there exists a time τ' such that for all $t \ge \tau'$

$$(1+\rho)^{-1}t+a^{(1)} \le C_2^{(1)}(t) \le (1+\rho)^{-1}t+b^{(3)}$$

From these two relations it follows that in execution e_1, for any given D_{max}, there is some time t' such that for all $t \ge t'$, the deviation between the two correct logical clocks is greater than D_{max}, which violates the *agreement* condition.

This can be generalized to any system of $n \ge 2$ processes, where $n \le 2f$. Partition the processes into two sets P_1 and P_2, with not more than f processes in either set. By constructing executions similar to those above, we can prove that no synchronization algorithm can achieve *optimal accuracy* if $n \le 2f$. □

The authenticated algorithm of Figure 1 requires $n > 2f$ processes. By Theorem 3, this algorithm can be modified to achieve *optimal accuracy*. From Theorem 4, it follows that the modified algorithm is also optimal in the number of faults tolerated.

4. Synchronization without authentication

4.1. Simulating authenticated broadcasts

The proof of correctness and the analysis of the authenticated algorithm rely on the following properties of the message system:

P1. *(Correctness)* If at least $f+1$ correct processes broadcast $(round\ k)$ messages by time t, then every correct process accepts the message by time $t + t_{del}$.

P2. *(Unforgeability)* If no correct process broadcasts a $(round\ k)$ message by time t, then no correct process accepts the message by time t or earlier.

P3. *(Relay)* If a correct process accepts the message $(round\ k)$ at time t, then every correct process does so by time $t + t_{del}$.

As seen earlier, implementing authentication using digital signatures provides these three properties. However, the correctness of the algorithm does not depend on this particular implementation, and any other implementation providing these properties can be used instead. A broadcast primitive to simulate authentication is described in [Srik84]. By replacing authenticated broadcasts in the algorithm of Figure 1 with this primitive, we get a logically equivalent non-authenticated algorithm having the properties of the authenticated algorithm. However, the number of messages sent by correct processes is $O(n^3)$ per resynchronization.

We now modify this broadcast primitive to achieve the three properties described above at a cost of only $O(n^2)$ messages per resynchronization. The primitive is presented in Figure 2, and requires $n \geq 3f+1$. With this primitive, each broadcast now requires two phases of communication. Therefore, t_{del}, the upper bound on the time required for a message to be prepared by a process, sent to all processes and processed by the correct processes accepting it, must be re-evaluated. Let τ be the maximum transmission delay between any two processes. Then, $t_{del} \geq 2\tau$.

Theorem 5: The broadcast primitive achieves properties of *correctness, unforgeability* and *relay*. The number of messages sent by correct processes is $O(n^2)$ per resynchronization.

To broadcast a $(round\ k)$ message, a correct process sends $(init, round\ k)$ to all.

for each correct process:

  if received $(init, round\ k)$ from at least $f+1$ distinct processes
    → send $(echo, round\ k)$ to all;

  [] received $(echo, round\ k)$ from at least $f+1$ distinct processes
    → send $(echo, round\ k)$ to all;

  fi

  if received $(echo, round\ k)$ from at least $2f+1$ distinct processes
    → accept $(round\ k)$ **fi**

Figure 2. A broadcast primitive to achieve properties P1, P2 and P3.

Proof:

(Correctness): Since at least $f+1$ correct processes broadcast (*round* k) by time t, every correct process receives at least $f+1$ (*init*, *round* k) messages by time $t+\tau$ and sends (*echo*, *round* k). Hence, by time $t+2\tau$, every correct process receives at least $2f+1$ (*echo*, *round* k) messages. That is, every correct process accepts (*round* k) by time $t+t_{del}$.

(Unforgeability): Since no correct process sends an (*init*, *round* k) message by time t, a correct process could have received (*init*, *round* k) messages from at most f processes and (*echo*, *round* k) messages from at most f processes. Thus, no correct process sends an (*echo*, *round* k) message by time t. Hence, no correct process accepts (*round* k) by time t.

(Relay): Since a correct process accepts (*round* k) at time t, it must have received at least $2f+1$ (*echo*, *round* k) messages. Every correct process receives at least $f+1$ of these within another τ and sends an (*echo*, *round* k) if it has not already done so. Hence, by $t+2\tau$ (i.e. by $t+t_{del}$), every correct process accepts a (*round* k) message.

Since each correct process sends at most 2 messages for each resynchronization round (an *init* and an *echo*), the total number of messages sent by correct processes is $O(n^2)$ per round.

□

4.2. A non-authenticated algorithm for clock synchronization

Replacing signed communication with our broadcast primitive extends the synchronization algorithm of Figure 1 to one for systems without authentication. The relay property of the primitive implies that we need not explicitly relay messages since the primitive does this automatically. Since the primitive requires $n>3f$, the non-authenticated algorithm also has this limit on the number of faulty processes. It has been shown in [Dole84] that if authentication is not available, and if there are no bounds on the rate at which faulty processes can generate messages, then synchronization is impossible unless $n>3f$.

As in Section 2, we assume that clocks are initially synchronized such that at $ready^1$, all correct processes are using C^0, and these clocks are at most $D_{\max}$ apart. The non-authenticated algorithm is described in Figure 3.

Theorem 6: The non-authenticated algorithm in Figure 3 achieves *agreement* and *accuracy*. Correct processes send $O(n^2)$ messages per resynchronization.

Proof: By properties P1 to P3 of the primitive of Figure 2, it is easy to see that the proofs of Lemmas 1 to 10 and Theorem 1 hold. Also, by Theorem 5, correct processes send $O(n^2)$ messages for each resynchronization round. □

Thus, the number of messages sent by correct processes for each resynchronization is comparable to that in [Lund84].

```
cobegin
  if C^{k-1}(t) = kP                       /* ready to start C^k */
      → broadcast (round k) fi             /* using the primitive in Figure 2 */
//
  if accepted the message (round k)        /* according to the primitive */
      → C^k(t) := kP + α  fi               /* start C^k */
coend
```

Figure 3. A non-authenticated algorithm for clock synchronization for process p for round k.

In Section 3.3, we showed how the authenticated algorithm could be modified to achieve *optimal accuracy*. Translating this modified algorithm with our broadcast primitive results in a non-authenticated algorithm that achieves *optimal accuracy*.

5. Initialization and integration

The algorithms presented in the previous sections can be used, with simple modifications, to achieve initial synchronization and to integrate new processes into the network.

Here we show how processes start their 0^{th} clocks close to each other. A process decides, independently, that it is time to start clock C^0 and broadcasts a *round* 0 message. On accepting a (*round* 0) message at real time t, it starts C^0 by setting $C^0(t)=\alpha$. The number of processes required, and the rules for accepting messages are as described in Sections 2 and 4, for the authenticated and non-authenticated systems, respectively. Since the authenticated and non-authenticated algorithms are equivalent, we illustrate only the non-authenticated version here (Figure 4).

It is easy to see that all processes start C^0 within t_{del} of each other. Also no correct process starts C^0 until at least one correct process is ready to do so. Once they have started C^0, processes run the resynchronization algorithm. At $ready^1$, which by definition is the time when a correct process first sends a (*round* 1) message, every correct logical clock reads P or

```
broadcast (round 0);                        /* using the primitive in Figure 2 */

if accepted the message (round 0)           /* according to the primitive */
    → C^0(t) := α  fi                       /* start C^0 */
```

Figure 4. A non-authenticated algorithm for achieving initial synchronization.

less. That is, every correct process is using C^0. By proofs similar to those in Lemmas 2 and 6, it can be seen that at $ready^1$, correct clocks are no more than $D_{\max}$ apart. Thus, this algorithm justifies assumptions S1 and S2 for $k=1$ in the proof of Lemma 8.

We now describe how a process joins a system of synchronized clocks. This could be used by new processes to enter the system, or by processes which have become unsynchronized (possibly due to failures) to re-establish synchronization with the rest of the system. The algorithms are based on the idea in [Lund84], modified to the context of our algorithms.

When a process p wishes to join the system, it sends a message (*joining*) to the processes already in the system. It then receives messages from these processes and determines the number i of the round being executed. Since p could have started this algorithm in the middle of a resynchronization period, it waits for resynchronization period $i+1$ and starts its logical clock C^{i+1} when it accepts a (*round* $i+1$) message. It is easy to prove that its clock is now synchronized with respect to the clocks already in the system. Process p now begins to run the resynchronization algorithm described earlier. We present only the non-authenticated version in Figure 5. This algorithm can also be modified as described in Section 3.3 to ensure that *optimal accuracy* is achieved.

This integration scheme prevents a (possibly faulty) process joining the system from affecting the correct processes already in the system. Hence, we prefer this "passive" scheme to that presented in [Halp84].

6. Restricted models of failure

In the preceding sections, we have assumed that faulty processes can exhibit arbitrary behavior. Fault-tolerant algorithms have also been studied under simpler, more restrictive models of failure. It is likely that in certain applications, faults are not as arbitrary as we have assumed so far. In such cases, developing algorithms for the simpler model of failure could result in easier and less expensive solutions.

The most benign type of failure is that of *crash* faults, where processes fail by just stopping [Lamp82, Hadz84]. Less restrictive models are *omission*, where faulty processes

---

send (*joining*) to all processes;
accept a (*round* i) message for some i;

if accepted the message (*round* $i+1$) /* wait for round $i+1$ */
  $\rightarrow C^{i+1}(t) := (i+1)P+\alpha$ **fi** /* start C^{i+1} */

Figure 5. A non-authenticated algorithm used by a process to join the system.

---

occasionally fail to send messages [Hadz84], or *sr-omission*, where faulty processes fail to send or receive messages [Perr84]. In this section, we show how the algorithms developed so far can be adapted to these models.

The algorithm of Figure 1 was shown to overcome arbitrary failures. The proof relied on an authenticated message system providing the properties P1 to P3. Consider systems with sr-omission failures, where a process is faulty either because it occasionally fails to send or receive messages, or because its physical clock does not satisfy assumption A1. For such systems, we can achieve properties P1 to P3 without authentication, using the broadcast primitive of Figure 6. With this broadcast primitive, the algorithm of Figure 3 is a synchronization algorithm for systems with sr-omission faults. Since crash faults and omission faults are a proper subset of sr-omission faults, the algorithm of Figure 3 can also tolerate these faults. As explained in Section 3.3, this algorithm is easily modified to achieve *optimal accuracy*. The primitive in Figure 6 requires $n > 2f$ processes and $t_{del} = \tau$. In contrast, the primitive of Figure 2 requires $n > 3f$ processes and $t_{del} = 2\tau$, but it overcomes arbitrary failures.

The lower bound proofs of Theorem 2 and Theorem 4 do not make any assumptions on the behavior of faulty processes. In fact, we only required that the clocks of faulty processes run at arbitrary rates with respect to real time. Therefore, both lower bounds hold even for crash faults. Thus, our synchronization algorithm is optimal in the number of faults that can be tolerated for all the models of failure we consider.

Initial synchronization and integration of new clocks are achieved as in previous sections.

To broadcast a (*round k*) message, a correct process sends (*init*, *round k*) to all.

for each correct process:

```
if  received (init, round k) from at least f+1 distinct processes
      → accept (round k);
        send (echo, round k) to all;
[] received (echo, round k) from any process
      → accept (round k);
        send (echo, round k) to all;
fi
```

Figure 6. A broadcast primitive to achieve properties P1, P2 and P3 for a system with sr-omission failures.

---

7. Discussion

The requirements of synchronization can also be stated as follows [Halp84, Dole84]: there exist constants $d_{\min}, P, D_{\max}$ and ADJ, such that clocks are resynchronized at logical times that are multiples of P, and for all correct clocks i and j and for all $k \geq 1$:

C1. $\forall t \in [end^k, end^{k+1}]$

$$\left| C_i^k(t) - C_j^k(t) \right| \leq D_{\max}$$

C2. If C_i^k is started at time t, then

$$0 \leq C_i^k(t) - C_i^{k-1}(t) \leq ADJ$$

C3. $0 \leq end^k - beg^k \leq d_{\min}$

These conditions assert that the maximum deviation between correct clocks is bounded, the amount by which clocks are re-adjusted is bounded, and the size of a resynchronization period is small. Our algorithms satisfy these conditions. Lemmas 1 and 6 show that conditions C1 and C3 are satisfied. From Lemma 4, we see that clocks are never set back. It is easy to show that the maximum adjustment made is $\alpha + D_{\max}$. Hence, by setting $ADJ = \alpha + D_{\max}$, condition C2 is also met.

A feature of our algorithm is that $d_{\min}, P$, and ADJ depend only on the system parameters ρ and t_{del} , and on the constraint $D_{\max}$. In the authenticated algorithm in [Halp84], the adjustment ADJ is proportional to the number of faulty processors. Our solution does not use averaging, and for the non-authenticated case, given $D_{\max}$, the maximum permitted deviation between correct clocks, our algorithm needs about half as many resynchronizations as in the best previous result [Lund84]. The minimum value of $D_{\max}$ that our algorithm can achieve depends only on ρ and t_{del}. In [Lamp85], the minimum $D_{\max}$ possible is proportional to the number of processes in the system.

In the preceding sections, we have assumed a completely connected network. This assumption can be relaxed using well-known techniques. For an authenticated system, node connectivity of $f+1$ is sufficient. This ensures that there is at least one fault-free path between every pair of correct processes. As in [Halp84], by defining t_{del} to be the maximum time to transmit a message between correct processes along at least one fault-free path in the network, the results of Section 2 hold.

Similarly, a non-authenticated system with node connectivity of $2f+1$ provides at least $f+1$ distinct fault-free paths between each pair of correct processes. Define t_{del} to be twice the maximum time taken for a message to be relayed along $f+1$ fault-free paths. Again, the results proved earlier for the non-authenticated system hold.

8. Conclusion

In this paper, we have presented a simple, efficient and unified solution to the problems of synchronizing clocks, initializing these clocks, and integrating new clocks, for systems with different types of failures: crash, omission, and arbitrary failures, with and without message authentication. This solution was derived with the help of the methodology described in [Srik84].

This is the first known solution that achieves *optimal accuracy*, i.e., the accuracy of synchronized clocks (with respect to real time) is as good as that specified for the underlying hardware clocks. The algorithms presented are also optimal with respect to the number of faulty processes that can be tolerated to achieve this accuracy.

Acknowledgements

We are grateful to Abha Moitra and Jennifer Lundelius for their helpful comments and suggestions.

References

Dole84 D. Dolev, J.Y. Halpern, and R. Strong, On the possibility and impossibility of achieving clock synchronization, *Proceedings Sixteenth Annual ACM STOC,* Washington D.C., pp. 504-511, April 1984.

Halp84 J.Y. Halpern, B. Simons, R. Strong, and D. Dolev, Fault-tolerant clock synchronization, *Proceedings Third Annual ACM Symposium on Principles of Distributed Computing,* Vancouver, Canada, pp. 89-102, Aug. 1984.

Hadz83 V. Hadzilacos, Byzantine Agreement under restricted types of failures (not telling the truth is different from telling lies), Tech. Rep. 19-83, Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts, June 1983.

Lamp82 L. Lamport and M. Fischer, Byzantine Generals and Transaction Commit protocols, Opus 62, SRI International, April 1982.

Lamp85 L. Lamport and P.M. Melliar-Smith, Synchronizing clocks in the presence of faults, *Journal of the ACM,* vol. 32, No. 1, pp. 52-78, Jan. 1985.

Lund84 J. Lundelius and N. Lynch, A new fault-tolerant algorithm for clock synchronization, *Proceedings Third Annual ACM Symposium on Principles of Distributed Computing,* Vancouver, Canada, pp. 75-88, Aug. 1984.

Perr84 K. J. Perry and S. Toueg, Distributed Agreement in the Presence of Processor and Communication Faults, Tech. Rep. 84-610, Department of Computer Science, Cornell University, Ithaca, New York, May 1984. To appear in the *IEEE Transactions on Software Engineering*, November 1985.

Srik84 T.K. Srikanth and S. Toueg, Simulating authenticated broadcasts to derive simple fault-tolerant algorithms, Tech. Rep. 84-623, Department of Computer Science, Cornell University, Ithaca, New York, July 1984.