INFORMATION AND CONTROL **62**, 190–204 (1984)

An Upper and Lower Bound for Clock Synchronization*

JENNIFER LUNDELIUS AND NANCY LYNCH

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

The problem of synchronizing clocks of processes in a fully connected network is considered. It is proved that, even if the clocks all run at the same rate as real time and there are no failures, an uncertainty of ε in the message delivery time makes it impossible to synchronize the clocks of n processes any more closely than $\varepsilon(1 - 1/n)$. A simple algorithm is given that achieves this bound. © 1984 Academic Press, Inc.

1. INTRODUCTION

Keeping the local clocks of processes synchronized in a distributed system is important in many applications and is an interesting problem in its own right. In order to be practical, algorithms to synchronize clocks should be able to tolerate process failures, clock drift, and varying message delivery times. However, these conditions complicate the design and analysis of algorithms.

In this paper, we consider a simple special case of the general clock synchronization problem. Namely, we assume that clocks run at a perfect rate and that there are no failures. However, clocks initially have arbitrary values, and there is an uncertainty of ε in the message delivery time. For this case, once the clocks are synchronized, they will remain synchronized, so the only problem is to synchronize them in the first place.

We show that, even under these simplifying assumptions, no algorithm can synchronize clocks exactly. More precisely, we show that $\varepsilon(1 - 1/n)$ is a lower bound on how closely the clocks of n processes can be synchronized in this case. Since these are strong assumptions, this lower bound also holds for the more realistic case in which clocks drift and arbitrary faults occur. We show that the bound of $\varepsilon(1 - 1/n)$ is tight for the simplified case, by describing a simple algorithm that achieves this bound.

* This work was supported in part by the NSF under Grant MCS83-06854, 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.

0019-9958/84 $3.00

The problem of synchronizing clocks in a distributed system has been a topic of considerable research interest recently. Several algorithms have appeared in the literature (Halpern, Simons, and Strong, 1983; Halpern, Simons, Strong, and Dolev, 1984; Lamport, 1978; Lamport and Melliar-Smith, 1984; Lundelius and Lynch, 1984; Marzullo, 1983), each working under different assumptions. Dolev, Halpern, and Strong (1984) show that it is impossible to synchronize clocks if one third or more of the processes are subject to Byzantine failures. They also demonstrate a lower bound similar to ours (proved independently), but characterizing the closeness of synchronization obtainable along the real time axis. That is, they prove a lower bound on how close the real times can be when two processes' clocks have the same value, whereas our result is a lower bound on how close the clock values can be at the same real time.

The remainder of the paper is organized as follows. Section 2 contains a description of the properties we require of our system model, and a statement of the clock synchronization problem of this paper. Section 3 contains the lower bound result, and Section 4 contains the corresponding upper bound. We conclude with an open question in Section 5.

2. The Clock Synchronization Problem

2.1. *Systems of Processes with Clocks*

One way of presenting our results would be by using a specific formal model for systems of processes with clocks. However, the results of this paper are not dependent on the precise details of a particular model. Therefore, we do not give a complete description of a formal model in this paper; rather, we just state the properties which we require of such a model. We refer the interested reader to Lundelius and Lynch (1984) for a detailed development of a particular model for systems of processes with clocks; also, preliminary versions of the results of the present paper are given in terms of such a model in Lundelius (1984).

The system is assumed to consist of *n processes*, located at the vertices of a complete communication graph. All processes are assumed to know the size and topology of the network. Each process has a local "physical clock," whose value it can read. Processes communicate by sending and receiving messages.

We do not make many explicit assumptions about the form of a process. We presume that a process can be modelled as some kind of automaton, having a state set, including initial and final states, and a transition relation, which defines the algorithm to be executed. However, processes might be deterministic or nondeterministic. They might be assumed to have significant or insignificant local processing time. They might buffer incoming messages until they are ready to process them, or they might process incoming messages immediately. They might take steps only upon receipt of a message, or also upon discovering that their physical clocks have reached certain values, or at arbitrary times. Many other variations are possible, and our results will hold equally well for all of these cases.

We introduce some notation and definitions. Let P be a set of n processes. A *clock* is a monotone increasing function from $\mathbb{R}$ (real time) to $\mathbb{R}$ (clock time). In this paper, we assume that clocks do not drift; thus, we assume that all clock functions have derivative exactly 1 everywhere. A *system of processes with clocks* (or simply a *system*), denoted by $(P, \mathscr{C})$, is a set of processes P together with a set of clocks $\mathscr{C} = \{C_p\}$, one for each p in P. Clock C_p is called p's *physical clock.*

Each process' physical clock is assumed to be a fixed function, i.e., it cannot be modified by the process. We assume that processes do not have access to the real time; each process obtains its only information about time from its physical clock. Thus, a process' physical clock value might be used in its transition relation, but the real time cannot be so used. By modelling the clocks separately from the processes, we can study the effect of using different clock functions with the same set of processes.

2.2. *Executions*

In this subsection, we define the "executions" of a system of processes with clocks. We begin by defining executions for individual processes. The *events* which can occur at a process include the arrival of messages from other processes, as well as any significant events internal to the process. These events may cause the process to send messages to other processes. An *action* describes the changes made by a particular event to the process' state. An *execution* of process p with clock C is a partial mapping from $\mathbb{R}$ (real time) to actions; the action for a given time describes the changes to p which occur at that time. Process executions are assumed to satisfy certain constraints, as given by the process model and the particular process definition.

An *execution* for a system $(P, \mathscr{C})$ of processes with clocks is a set of process executions, one for each process p in P, with clock C_p in $\mathscr{C}$, together with a one-to-one correspondence between the messages sent by p to q and the messages received by q from p, for any processes p and q. We use the message correspondence to define the *delay* of any message in a system execution, in the obvious way. For each system execution e, define *last-step* (e) to be the earliest time in e at which all processes are in final states. If there is no such time, then last-step (e) is undefined.

2.3. *Views and Equivalence*

As we have already stated, we are assuming that the processes do not have access to the real time, but only to their physical clock time. In the lower bound proof, we will consider different system executions that are indistinguishable to the processes because the events occur at the same physical clock times, although they might occur at different real times.

Thus, we define the *view* of any process p in any process execution e (for p with clock C), to be the actions in e, together with their physical clock times of occurrence. The real times of occurrence are not represented in the view. The notion of a view allows us to define a natural notion of equivalence for process executions. Define two process executions, one of process p with clock C and the other of process p with clock C', to be *equivalent* provided that the view of p is the same in both executions. We extend this definition to a definition of equivalence for system executions. Define two system executions, execution e of system $(P, \mathscr{C})$ and execution e' of $(P, \mathscr{C}')$, to be *equivalent* provided that for each process p, the component process executions for p in e and e' are equivalent. Thus, the executions are indistinguishable to the processes. Only an outside observer who has access to the real time can tell them apart.

2.4. *Shifting*

We introduce the notion of "shifting," both for a system execution and for a set of clocks. Shifting a system execution by some amount, relative to p, means modifying p's process execution so that every action for p occurs that amount earlier in real time. Shifting a set of clocks by some amount, relative to a process p, means adding that amount to the function that defines p's clock. We make assumptions which insure that, if an execution and a set of physical clocks are both shifted by the same amount relative to the same process, the resulting execution is equivalent to the original one. No process can tell the difference, because the change in the time of occurrence of actions in the execution is compensated for by the change in the physical clock.

We begin by defining a shift of a process execution and of a single clock. Given execution e of process p with clock C, and real number ζ, a new execution $e' = shift(e, \zeta)$ is defined by $e'(t) = e(t + \zeta)$ for all t. All actions are shifted earlier in e' by ζ if ζ is positive, and later by $-\zeta$ if ζ is negative. Given a clock C and real number ζ, a new clock $C' = shift(C, \zeta)$ is defined by $C'(t) = C(t) + \zeta$ for all t. The clock is shifted forward by ζ if ζ is positive, and backward by $-\zeta$ if ζ is negative.

We make the following important assumption.

AXIOM 1. *Let e be an execution of process p with clock C, and let ζ be a* *real number. Let $C' = shift(C, \zeta)$. Then $shift(e, \zeta)$ is an execution of p with clock C'.*

That is, if a process execution and physical clock are modified in corresponding ways, the result is also an execution. It is easy to see that this resulting execution must be equivalent to the original execution.

Now we define a shift of a system execution and of a set of clocks. Given execution e of system $(P, \mathscr{C})$, and real number ζ, a new execution $e' = shift(e, p, \zeta)$ is defined by replacing p's process execution in e, e_p, by shift(e_p, ζ), and by retaining the same correspondence between sends and receives of messages. (Technically, the correspondence is redefined so that a pairing in e that involves the event for p at time t, in e' involves the event for p at time $t - \zeta$).) All actions for process p are shifted by ζ, but no other actions are altered. Given a set of clocks $\mathscr{C} = \{C_q\}_{q \in P}$, and real number ζ, a new set of clocks $\mathscr{C}' = shift(\mathscr{C}, p, \zeta)$, is defined by replacing clock C_p by clock shift(C_p, ζ). Process p's clock is shifted forward by ζ, but no other clocks are altered.

LEMMA 1. *Let e be an execution of system $(P, \mathscr{C})$, p a process and ζ a real number. Let $\mathscr{C}' = shift(\mathscr{C}, p, \zeta)$ and $e' = shift(e, p, \zeta)$. Then e' is an execution of $(P, \mathscr{C}')$, and e' is equivalent to e.*

Proof. The result follows immediately from the definition of a system execution, together with Axiom 1 and the immediately following remarks. ∎

The following lemma quantifies how message delays change when a system execution is shifted.

LEMMA 2. *Let e be an execution of system $(P, \mathscr{C})$, p a process, ζ real number. Let $\mathscr{C}' = shift(\mathscr{C}, p, \zeta)$ and $e' = shift(e, p, \zeta)$. Then when the obvious correspondence is made between messages in e and in e', all messages have the same delay in e' as in e, with the following two exceptions. If q is any process other than p, then*

(a) *the delay of any message from q to p is ζ less in e' than in e, and*

(b) *the delay of any message from p to q is ζ greater in e' than in e.*

Proof. Without loss of generality, assume ζ is nonnegative. Since all events for p happen ζ earlier in e' than in e, and since the correspondence between sends and receives is updated appropriately, messages are received

ζ earlier (causing ζ less delay), and are sent ζ earlier (causing ζ greater delay). ▌

2.5. *Admissible Executions*

For the remainder of the paper, fix nonnegative values ε, μ, and ν such that $\nu - \mu = \varepsilon$. We say that a system execution e is *admissible* provided that for every p and q, every message in e from p to q has its delay in the range $[\mu, \nu]$. Thus, μ is the smallest message delay, ν is the largest delay, and the difference between them, ε, is the *message uncertainty.*

We note that our results would hold with almost identical proofs in the case where μ and ν differ from link to link, as long as ε is the same. The restriction to uniform μ and ν is made only for notational simplicity.

2.6. *Problem Statement*

Now we describe the particular clock synchronization problem which is considered in this paper. Assume that the system model is as described so far in this section. We consider only admissible executions, and we assume further that the processes have knowledge of the message delay bounds μ and ν.

The processes are supposed to establish synchronization of their "local times." These local times are not the values of the physical clocks, since we assume that the physical clocks cannot be reset by the processes. Rather, each process obtains its notion of the local time by adding the value in a particular local variable CORR to the physical clock time. The process is able to modify the value in its CORR variable, so that during an execution, p's local variable CORR can take on different values. We assume that the value of CORR is 0 in any initial state, and cannot be changed after a process enters a final state. For a particular execution, we define a function $\mathrm{CORR}_p(t)$, giving the value of p's variable CORR at time t. Then, for a particular execution, we define the *local time* for p to be the function L_p, which is given by $C_p + \mathrm{CORR}_p$.

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

Since an algorithm is coded into the transition function for a process, P is all that is needed to specify an algorithm. A clock synchronization algorithm P is said to *synchronize to within* γ if the algorithm terminates (i.e., all processes eventually enter final states), and after it terminates, the processes' local times differ by no more than γ. More precisely, we require that every admissible execution e for $(P, \mathscr{C})$, for any set of clocks $\mathscr{C}$, satisfies the following conditions:

(a) Termination. All processes eventually enter final states. Thus, last-step(e) is defined.

(b) Agreement. $|L_p(t) - L_q(t)| \leqslant \gamma$ for any processes p and q and time $t \geqslant$ last-step(e).

3. Lower Bound

In this section we show that no algorithm can synchronize n processes' clocks any more closely than $\varepsilon(1-1/n)$. The main idea of the proof is that different executions can be constructed that look the same to the processes but that result in different local times. We consider an arbitrary algorithm P that synchronizes clocks to within γ. We begin with an admissible execution of P that has a particular pattern of message delays, and then alter this execution by judicious shifting so that the resulting message delays are still within the allowable range (i.e., the result is another admissible execution), and so that no process can tell the difference (i.e., the old and new executions are equivalent). The equivalence implies an inequality concerning γ. By constructing n equivalent executions in this manner, n inequalities concerning γ are obtained. Solving the inequalities for γ produces the claimed lower bound.

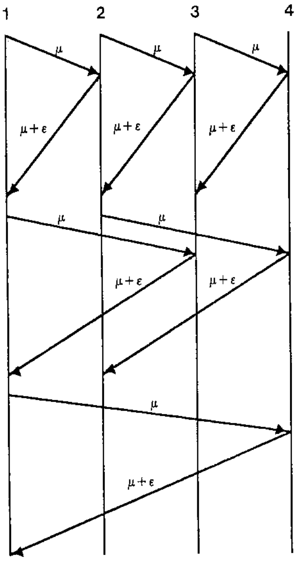
Theorem 3. *No clock synchronization algorithm can synchronize a system of n processes to within γ, for any $\gamma < \varepsilon(1-1/n)$.*

Proof. Fix a set of processes P that synchronizes to within γ. We will show that $\gamma \geqslant \varepsilon(1-1/n)$.

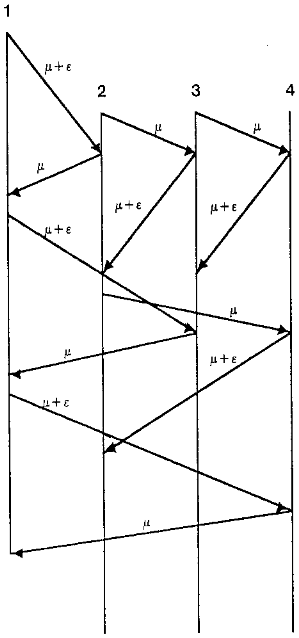
Let P consist of processes 1 through n. We construct a sequence of systems $\mathscr{S}^i = (P, \mathscr{C}^i)$, for $1 \leqslant i \leqslant n$, and a corresponding sequence of executions e^i for those systems. All of the executions e^i will be equivalent to each other, and all will be admissible. Furthermore, in e^i, all messages sent by process i will have delay μ and all messages received by i will have delay ν. The construction is carried out inductively on i.

Let $\mathscr{S}^1 = (P, \mathscr{C}^1)$, where $\mathscr{C}^1$ is an arbitrary set of clocks. Let e^1 be any execution of $\mathscr{S}^1$ in which all messages from process j to process k have delay exactly μ if $j < k$, and have delay exactly ν if $j > k$. That is, messages from processes to higher-numbered processes take the minimum delivery time, while messages from processes to lower-numbered processes take the maximum delivery time. Clearly, e^1 is admissible, all messages sent by process 1 have delay μ, and all messages received by process 1 have delay ν. (For the special case where $n = 4$, we represent the execution e^1 as in Fig. 1. There is a vertical line for each of the four processes. All the messages from j to k have the delay that labels the arrow from j to k.)

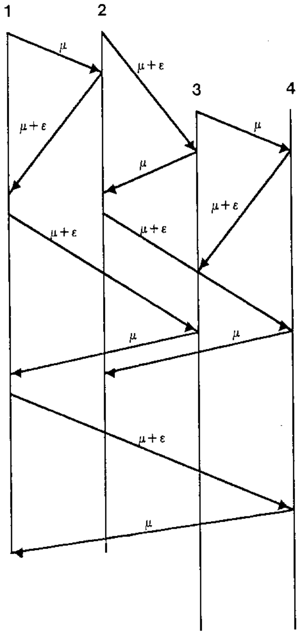
1



2



3



4

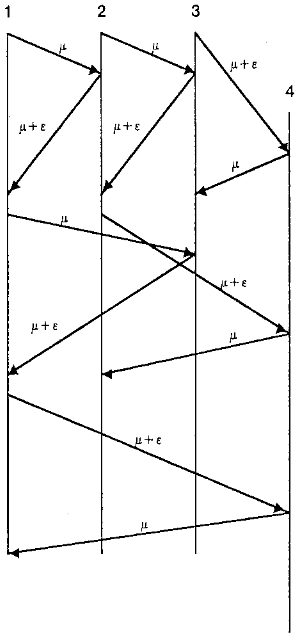


FIG. 1. Message delays for execution e^1 in the case $n=4$.

FIG. 2. Message delays for execution e^2 in the case $n=4$.

FIG. 3. Message delays for execution e^3 in the case $n=4$.

FIG. 4. Message delays for execution e^4 in the case $n=4$.

Now assume that $\mathscr{S}^{i-1}$ and e^{i-1} have been constructed for $2 \leqslant i \leqslant n$, and, furthermore, that e^{i-1} is admissible, and that, in e^{i-1}, all messages sent by process $i-1$ have delay μ and all messages received by $i-1$ have delay v. We construct $\mathscr{S}^i$ and e^i. Let $\mathscr{C}^i = \text{shift}(\mathscr{C}^{i-1}, i-1, \varepsilon)$ and $\mathscr{S}^i = (P, \mathscr{C}^i)$. Let $e^i = \text{shift}(e^{i-1}, i-1, \varepsilon)$. Thus, the ith execution is obtained from the $(i-1)$th execution by shifting the execution and set of clocks by ε relative to process $i-1$. (For the case of $n=4$, the three executions e^2, e^3, and e^4 are depicted in Figures 2, 3, and 4.)

By Lemma 1 and the inductive hypothesis, e^i is an execution of $(P, \mathscr{C}^i)$, and is equivalent to e^{i-1}. We now argue that e^i is admissible. By Lemma 2, the only changes between e^{i-1} and e^i are for messages involving process $i-1$. Messages received by $i-1$ take ε less time, so they have delay $v-\varepsilon=\mu$; messages sent by $i-1$ take ε more time, so they have delay $\mu+\varepsilon=v$. These delays are in the specified range.

The last part of the induction is showing that in e^i all messages received by process i have delay v and all messages sent by process i have delay μ. Messages to and from a higher-numbered process have delays as in e^1, i.e., μ and v, respectively. All lower-numbered processes have been shifted by ε, so the delays, which were originally μ (for receiving) and v (for sending) have become $\mu+\varepsilon=v$ and $v-\varepsilon=\mu$, respectively.

Since e^1 is an admissible execution, it must terminate; let $t_f=$ last-step(e^1). By equivalence, all the e^i terminate, and the direction of the shifts implies that they all terminate by time t_f.

Let $V_1,\ldots, V_n$ be the values for the respective processes' local times at real time t_f, in execution e^1. Since the algorithm is assumed to synchronize to within γ, all of these values are within γ of each other. In particular,

$$V_n \leqslant V_1 + \gamma.$$

Now consider e^i, $1 < i \leqslant n$. Since e^i is equivalent to e^1, the correction variable for any process p is the same in both executions at real time t_f. This fact, together with the definition of $\mathscr{C}^i$, implies that in e^i, process $i-1$'s local time at real time t_f is $V_{i-1}+\varepsilon$ and process i's local time at real time t_f is V_i. Since these values must be within γ of each other, we have

$$V_{i-1} \leqslant V_i + \gamma - \varepsilon.$$

Adding the n inequalities together and collecting terms, we have

$$\sum_{i=1\cdots n} V_i \leqslant \sum_{i=1\cdots n} V_i + n\gamma - (n-1)\,\varepsilon,$$

or

$$(n-1) \leqslant n\gamma.$$

In order for this inequality to hold, it must be the case that $\gamma \geqslant \varepsilon(1-1/n)$. ∎

4. Upper Bound

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

4.1. *Algorithm*

There is an extremely simple algorithm that achieves the closest possible synchronization. Define δ to be $(\mu+\nu)/2$, the median message delay. As soon as each process p awakens, it sends its local time in a message to the remaining processes and waits to receive a similar message from every other process. Immediately upon receiving such a message, say from q, p estimates q's current local time by adding δ to the value received. Then p computes the difference between its estimate of q's local time and its own current local time. After receiving local times from all the other processes, p sets its correction variable to the average of the estimated differences (including 0 for the difference between p and itself).

We describe this algorithm below in pseudo-code. The particular language used can be translated unambiguously into the formal model of Lundelius and Lynch (1984); we refer the reader to that paper for more details. For this paper, we do not require the complete generality; thus, we just describe the meaning of the single program below.

The algorithm is interrupt-driven, where an interrupt can be either the arrival of a message or the arrival of a special START signal from the outside world. A *beginstep*(u) statement indicates the beginning of a step of the process, triggered by interrupt u. The step of the process continues (indivisibly), executing statements of the code just until the next *endstep* statement is reached. Then the process suspends execution until another interrupt arrives.

We assume that the state of a process consists of values for all the local variables, DIFF, SUM, RESPONSES, and CORR, together with a location counter which indicates the next beginstep statement (if any) to be executed. The initial state of a process consists of the value 0 for all the local variables, and the location counter positioned at the first beginstep statement of the program. Final states are those in which the location counter is at the end of the code. A step of the process involves receiving an interrupt, reading the local physical clock, carrying out some local computation (which can read and modify the variables and location counter in the process state), and perhaps sending some messages. NOW indicates the current local time.

Code for Process p:

```
beginstep($u$)
send (NOW) to all $q \neq p$

do forever
    if $u =$ message $V$ from process $q$ then
        DIFF := $V + \delta -$ NOW
        SUM := SUM + DIFF
        RESPONSES := RESPONSES + 1
        endif
    if RESPONSES $= n - 1$ then exit endif
    endstep
    beginstep($u$)
    enddo

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

For the remainder of the paper, fix P to be a set of n processes, each running the preceding code.

4.2. *Correctness*

We will show that any admissible execution e of the algorithm synchronizes to within γ, where γ is fixed for this section as $\varepsilon(1 - 1/n)$. The upper bound is not quite as strange as it might look at first glance. It can be rewritten as $(2(\varepsilon/2) + (n-2)\,\varepsilon)/n$, which is the average of the possible discrepancies between the estimates two particular process p and q can make, for the values of the physical clocks of all the processes. Processes p and q can agree on a clock value for p (or for q) to within accuracy at most $\varepsilon/2$ (giving the $2(\varepsilon/2)$ term, and can agree on a clock value for any other process r to within accuracy at most ε (giving the $(n-2)\,\varepsilon$ term). Then the possible discrepancies are averaged, so the sum is divided by n.

We now give a careful analysis. Fix $\mathscr{C}$ to be an arbitrary set of physical clocks; we must show that $\mathscr{S} = (P, \mathscr{C})$ synchronizes to within γ. First, we define Δ_{pq}, the actual difference between the physical clocks of p and q, to be $C_p - C_q$. Since there is no drift in the clock rates, this difference is a well-defined constant. Moreover, note the following.

Lemma 4. *For any processes p, q, and r,*

(a) $\Delta_{pp} = 0$,

(b) $\Delta_{pq} = -\Delta_{qp}$,

(c) $\Delta_{pq} = \Delta_{pr} + \Delta_{rq}$.

Proof. Immediate from the definition of Δ. ▌

Next, we define D_{pq}, the estimated difference between the physical clocks of p and q, as estimated by q. For $p \neq q$, let D_{pq} be the value of process q's local variable DIFF immediately after process p's message is handled by process q. It is easy to see that $D_{pq} = C_p(t) + \delta - C_q(t')$, where local time $L_p(t) = C_p(t)$ is sent by p at real time t and received by q at real time t'. Let $D_{pp} = 0$. We relate the estimates D to the actual differences Δ.

LEMMA 5. *Let p and q be processes. Then* $|D_{pq} - \Delta_{pq}| \leqslant \varepsilon/2$.

Proof. Suppose at real time t, p sends the value $C_p(t)$, which is received by q at real time t'. Then

$$\begin{aligned}
|D_{pq} - \Delta_{pq}| &= |C_p(t) + \delta - C_q(t') - \Delta_{pq}| \\
&= |C_q(t) + \Delta_{pq} + \delta - C_q(t') - \Delta_{pq}|, \qquad \text{by definition of } \Delta_{pq}, \\
&= |C_q(t) + \delta - C_q(t')| \\
&= |\delta - (C_q(t') - C_q(t))| \\
&= |\delta - (t' - t)|, \qquad \text{since the rate of clock } C_q \text{ is } 1, \\
&\leqslant \varepsilon/2, \qquad \text{since } \delta - \varepsilon/2 \leqslant t' - t \leqslant \delta + \varepsilon/2. \quad ▌
\end{aligned}$$

The next lemma concerns the relationships between two processes' estimated differences and the actual differences.

LEMMA 6. *Let p, q, and r be processes. Then*

(a) $|(D_{pq} - D_{pr}) - \Delta_{rq}| \leqslant \varepsilon$,

(b) $|(D_{pp} - D_{pr}) - \Delta_{rp}| \leqslant \varepsilon/2$,

(c) $|(D_{pq} - D_{pp}) - \Delta_{pq}| \leqslant \varepsilon/2$.

Proof.

(a)

$$\begin{aligned}
|(D_{pq} - D_{pr}) - \Delta_{rq}| &= |(D_{pq} - D_{pr}) - (\Delta_{pq} - \Delta_{pr})|, \qquad \text{by Lemma 4}, \\
&= |D_{pq} - \Delta_{pq}) - (D_{pr} - \Delta_{pr})| \\
&\leqslant |D_{pq} - \Delta_{pq}| + |D_{pr} - \Delta_{pr}| \\
&\leqslant \varepsilon, \qquad \text{by two applications of Lemma 5.}
\end{aligned}$$

(b)

$$
\begin{aligned}
|(D_{pp}-D_{pr})-\Delta_{rp}| &\leqslant |D_{pp}-\Delta_{pp}|+|D_{pr}-\Delta_{pr}|, && \text{as in part (a),}\\
&= 0+|D_{pr}-\Delta_{pr}|,\\
&\leqslant \varepsilon/2, && \text{by Lemma 5.}
\end{aligned}
$$

(c) is similar to (b), and is left to the reader. ▮

Here is the main result.

THEOREM 7 (Agreement). *Algorithm P guarantees clock synchronization to within* $\varepsilon(1-1/n)$.

Proof. Fix a set of clocks $\mathscr{C}$, and let $\mathscr{S}=(P,\mathscr{C})$. We must show that for any admissible execution e of $\mathscr{S}$, any two processes p and q, and any time t after last-step(e),

$$|L_p(t)-L_q(t)| \leqslant \varepsilon(1-1/n).$$

Now

$$
\begin{aligned}
|L_p(t)-L_q(t)| &= |(C_p(t)+\mathrm{CORR}_p(t))-(C_q(t)+\mathrm{CORR}_q(t))|\\
&= |\Delta_{pq}-(\mathrm{CORR}_q(t)-\mathrm{CORR}_p(t))|\\
&= |\Delta_{pq}-((1/n)\sum_{r\in P} D_{rq}-(1/n)\sum_{r\in P} D_{rp})|,\\
&\qquad \text{by the way the algorithm works,}\\
&= (1/n)\left|n\Delta_{pq}-\sum_{r\in P}(D_{rq}-D_{rp})\right|\\
&= (1/n)\left|\sum_{r\in P}(\Delta_{pq}-(D_{rq}-D_{rp}))\right|\\
&= (1/n)\left|\sum_{r\in P}((\Delta_{rq}-\Delta_{rp})-(D_{rq}-D_{rp}))\right|, \qquad \text{by Lemma 4,}\\
&\leqslant (1/n)\sum_{r\in P}|(\Delta_{rq}-\Delta_{rp})-(D_{rq}-D_{rp})|.
\end{aligned}
$$

Now, the summation consists of n terms, each of which can be bounded using Lemma 6. The two terms for $r=p$ and $r=q$ are each bounded by $\varepsilon/2$, while the other $n-2$ terms are each bounded by ε. Thus, the entire expression is

$$
\begin{aligned}
|L_p(t)-L_q(t)| &\leqslant (1/n)(2\varepsilon/2+(n-2)\,\varepsilon)\\
&= \varepsilon(1-1/n). \quad ▮
\end{aligned}
$$

4.3. *Validity*

There is one other property of the algorithm which is worth noting. Namely, it produces local times which are not very far from the values of the physical clocks of the processes. We make this condition more precise by defining a clock synchronization algorithm P to be *α-valid* provided that for every $\mathscr{C}$ and every admissible execution e for $(P, \mathscr{C})$, the following is true. For any process p, there exist processes q and r such that $C_q(t) - \alpha \leqslant L_p(t) \leqslant C_r(t) + \alpha$ for all times t after last-step(e).

THEOREM 8. *Algorithm P is $\varepsilon/2$-valid.*

Proof. Let e be an admissible execution for $(P, \mathscr{C})$, where $\mathscr{C}$ is any set of physical clocks. Let p be any process, and let t be any time after last-step(e). By definition, the value of CORR_p at time t is equal to the average, $(1/n) \sum_{q \in P} D_{qp}$. Then there exist processes q and r such that

$$D_{qp} \leqslant \mathrm{CORR}_p(t) \leqslant D_{rp}.$$

By applying Lemma 5 to each end of this inequality, we get

$$\Delta_{qp} - \varepsilon/2 \leqslant D_{qp} \leqslant \mathrm{CORR}_p(t) \leqslant D_{rp} \leqslant \Delta_{rp} + \varepsilon/2.$$

Thus, $C_p(t) + \Delta_{qp} - \varepsilon/2 \leqslant C_p(t) + \mathrm{CORR}_p(t) \leqslant C_p(t) + \Delta_{rp} + \varepsilon/2$, which together with the definition of Δ implies that

$$C_q(t) - \varepsilon/2 \leqslant L_p(t) \leqslant C_r(t) + \varepsilon/2. \quad \blacksquare$$

5. OPEN QUESTION

It would be interesting to know how the results of this paper generalize to arbitrary communication graphs rather than just complete graphs. Also, it would be interesting to consider what happens when there are different uncertainties for the message delys on the different links.

ACKNOWLEDGMENTS

We thank Brian Coan, Cynthia Dwork, Joe Halpern, and Michael Merritt for their suggestions concerning earlier versions of this paper. Michael, in particular, gave us many excellent suggestions for improving the presentation.

RECEIVED May 14, 1984; accepted December 1984

References

Dolev, D., Halpern, J., and Strong, R. (1984), On the possibility and impossibility of achieving clock synchronization, *in* "Proc. 16th Annu. ACM Symposium on Theory of Computation," ACM SIGACT, Washington, D.C., pp. 504-511.

Halpern, J., Simons, B., and Strong, R. (1983), "An Efficient Fault-tolerant Algorithm for Clock Synchronization," IBM Computer Science Research Report RJ 4094 (45492).

Halpern, J., Simons, B., Strong, R., and Dolev, D. (1984), Fault-tolerant clock synchronization, *in* "Proc. 3rd Annu. ACM Symposium on Principles of Distributed Computing," ACM SIGACT and SIGOPS, Vancouver, pp. 89–102.

Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system, *Comm. ACM* **21**, No. 7, 558–565.

Lamport, L., and Melliar-Smith, P. M. (1984), Byzantine clock synchronization, *in* "Proc. 3rd Annu. ACM Symposium on Principles of Distributed Computing," ACM SIGACT and SIGOPS, Vancouver, pp. 68–74.

Lundelius, J. (1984), "Synchronizing Clocks in a Distributed System," S.M. thesis, Massachusetts Institute of Technology.

Lundelius, J. and Lynch, N. (1984), A new fault-tolerant algorithm for clock synchronization, *in* "Proc. 3rd Annu. ACM Symposium on Principles of Distributed Computing," ACM SIGACT and SIGOPS, Vancouver, pp. 75–88.

Marzullo, K. (1983), "Loosely-coupled Distributed Services: A Distributed Time Service," Ph.D. dissertation, Stanford University.