JOURNAL OF COMPUTER AND SYSTEM SCIENCES 32, 230–250 (1986)

On the Possibility and Impossibility of Achieving Clock Synchronization*

DANNY DOLEV

Hebrew University, Givat Ram, 91904 Jerusalem, Israel

JOSEPH Y. HALPERN AND H. RAYMOND STRONG

IBM Research Laboratory, San Jose, California 95193

Received November 5, 1984

It is known that clock synchronization can be achieved in the presence of faulty processors as long as the nonfaulty processors are connected, provided that some authentication technique is used. Without authentication the number of faults that can be tolerated has been an open question. Here we show that if we restrict logical clocks to running within some linear functions of real time, then clock synchronization is impossible without authentication when one-third or more of the processors are faulty. We also provide a lower bound on the closeness to which simultaneity can be achieved in the network as a function of the transmission and processing delay properties of the network. © 1986 Academic Press, Inc.

1. INTRODUCTION

The problem of achieving clock synchronization in the presence of faults has attracted much attention recently [LM, Ma, HSSD, LL1]. In [HSSD] an algorithm is presented that uses authentication (the ability to generate unforgeable signatures) and achieves synchronization with arbitrarily many faulty processors or communication links, provided that correct processors are not disconnected. Lamport and Melliar-Smith [LM] and Lundelius and Lynch [LL1] present algorithms that do not require authentication, but work only if fewer than one-third of the processors are faulty. It is known that Byzantine agreement cannot be achieved without authentication if at least one-third of the processors are faulty [LSP, PSL]. Until now the corresponding question for clock synchronization has remained open. It has been conjectured that the answer would be the same [LM]. In his invited address at the PODC Symposium in Montreal, Leslie Lamport challenged his listeners to provide a proof of impossibility [La]. In this paper we provide proofs of both possibility and impossibility, and show how sensitive the proofs are to the precise definition of clock synchronization. For the specific question posed by

* A preliminary version of this paper appeared in the Proceedings of the Sixteenth Annual ACM Symposium of Theory of Computation, Washington, D.C., 1984, and as IBM RJ 4218.

0022-0000/86 $3.00

Lamport with respect to the model of [LM], we provide the expected proof of impossibility.

For simplicity, we assume that each processor has a *physical clock* or *duration timer D*, and a designated register TAR called the *time adjustment register*. Physical clocks of correct processors drift away from real time by at most a bounded rate. The physical clock is never altered by the processor, but the time adjustment register may be altered of a processor's internal operations, receipt of messages, or an indication from its physical clock that a specific amount of time has elapsed. A processor's logical clock time C is the sum of TAR and D. Roughly speaking, and algorithm achieves clock synchronization if at all times the logical clock times of all correct processors are only a bounded distance apart.

Note that there is a trivial algorithm for clock synchronization: namely, whenever a processor's logical clock reads some predetermined value T, it is reset (by adjusting TAR) to read 0. We can eliminate this trivial solution by requiring that the range of a processor's logical clock be unbounded. However, as we show in Theorem 1 below, there is still a clock synchronization algorithm where the range of every processor's logical clock is unbounded that does not require any message passing. In this algorithm, a processor's logical clock runs at a rate that is roughly the logarithm of that of its physical clock. We eliminate this solution by requiring that the logical clock stay within a *linear envelope* of the physical clock, i.e., by requiring that there be constants a, b, c, and d such that $aD(t)+b \leqslant C(t) \leqslant cD(t)+d$ for all times t. (This condition is satisfied by all the clock synchronization algorithms mentioned above.) In this case we can show that clock synchronization is not achievable with one-third or more faulty processors (Theorem 2).

Finally we consider (Theorems 4 and 5) the degree to which simultaneity can be achieved in a network. We show that for every network there is a lower bound Δ such that no algorithm for clock synchronization can ensure that the difference between the (real) times at which two correct processors read a given value on their clocks is less than Δ. Δ is a function of the uncertainty in the time required to transmit and process a message. Results of [HSSD] show that this bound can essentially be achieved. These theorems bound the degree to which coordination can be achieved in a given network. It we want two events to happen simultaneously in a network, the best we can do is to guarantee that they happen at times separated by no more than Δ.

2. The Model

Each processor is connected to others via *links*. We do not assume that our network is completely connected; all our results hold regardless of the network topology. Processors send messages to each other along the communication links, where where a message is just a word over some fixed alphabet. As in most previous papers, we assume that a processor can always tell from which immediate neighbor in the communication network a message has come. We also assume that

associated with a link from processor p to q are two numbers $H(p, q)$ and $L(p, q)$ that represent known upper and lower bounds on the time to transmit (and process) a message from p to q. More formally, we take a *communication network* N to be a triple (G, H, L), where $G = (V, E)$ is a connected directed graph, and H and L are functions from E to the nonnegative reals such that $H(p, q) \geq L(p, q)$. $N = (G, H, L)$ is a network with *uncertain message transmission times* if for each link (p, q) in N, $H(p, q) > L(p, q)$.

We assume the existence of a real time frame, not directly observable. As we mentioned in the Introduction, each processor has a physical clock D and a time adjustment register TAR. Both of these can be viewed as real-valued functions of real time. A *correct* physical clock is one that thas a bounded rate of drift from real time. More precisely, there exists a constant $R \geq 1$ such that a correct physical clock satisfies, for all real times $u > v$,

$$(1/R)(u - v) \leq D(u) - D(v) \leq R(u - v).$$

Note that this inequality implies that a correct physical clock is a continuous, monotonic function of real time.

A *correct processor* is one that behaves according to its algorithmic specifications and possesses a correct physical clock. No assumption is made about the behavior of a *faulty* (not correct) processor.

We view $\mathrm{TAR}(t)$ as a function which is continuous on left-open right-closed intervals, where moreover, for all t, the limit of $\mathrm{TAR}(t')$ as t' approaches t from the right exists. We denote this limit $\mathrm{TAR}(t^+)$. Intuitively, if TAR is reset at discrete times, $\mathrm{TAR}(t^+)$ is the value of TAR just after it has been reset at time t. In all our algorithms, TAR will in fact be reset only at discrete times, so that it will be constant on left-open right-closed intervals. Altering TAR at these discrete times corresponds to starting a new clock in the notation of [LM, LL1, HSSD]. By definition, $C(t) = D(t) + \mathrm{TAR}(t)$; we take $C(t^+) = D(t) + \mathrm{TAR}(t^+)$. We think of $C(t)$ as representing a processor's logical clock at real time t.

We assume that all processors start up with their logical clocks, physical clocks, and TAR all set to 0. We do not need to assume that all processors start up at the same time, but we will assume that the processors all start within a short time span. Moreover, we assume that there is some constant B, depending only on the network, such that physical clocks of correct processors differ by at most B at the first point when all correct processors are up. Thus, correct clocks start out being close together; the problem is to keep them close together. We do not use this assumption in any essential way in our proofs; it is just used to give some initial bound on the difference between logical clocks. Note that if we assume that a processor starts up either spontaneously or on receipt of a message from another processor, then there is a simple algorithm that shows we can take $B \leq R(n-1)H$, where n is the number of processors in the network, and $H = \max_{p,q} H(p, q)$, provided that any pair of correct processors is connected by a fault-free path of links and processors in the network. We proceed as follows: as soon as a processor

starts up, it sends a message to all its neighbors. Clearly, within real time $(n-1)H$, every correct processor has received a startup message. In this interval, at most $R(n-1)H$ has passed on the physical clock of any processor.

We assume that processors perform *actions* such as sending messages and updating internal variables such as TAR according to some algorithm. An algorithm, in turn, is a function both of the messages sent and received and of the physical clock reading. To make this precise, we define processor p's *message history* at real time t to consist of a finite sequence of tuples of the form $\langle q, m, T, y \rangle$ (intuitively, one tuple for each message that it has sent or received up to, but not including, time t), where q indicates that the message was sent to or received from processor q, m is the value of the message, T indicates that $D_p(t') = T$ at the real time t' $(<t)$ when the message left or arrived, and y is "sent" or "received." Note that the messages marked "sent" in p's message history are redundant, in the sense that they are determined by the algorithm run and the message input. We include them here so that it is clear from p's message history exactly what messages p sent, even without knowing what algorithm p is running. This will be convenient when we describe our "two-faced" algorithm below.

We use H_p to denote processor p's *message history function*; $H_p(t)$ is p's message history at time t. For the most part in this paper we concentrate on *deterministic* algorithms; an action performed by processor p at real time t according to a deterministic algorithm $\mathscr{A}$ is, by definition, a function of $D_p(t)$ and $H_p(t)$. We comment in the concluding section on the extent to which our results hold for probabilistic algorithms, where the actions performed by a processor at time t may also depend on the result of a random coin toss.

We define a *scenario* to be a specification of the functions D_p and H_p for each processor p in the network and of the algorithm run by each processor. A *run* of an algorithm $\mathscr{A}$ in a communication network N is just a scenario in which all the processors in N run $\mathscr{A}$. We will say that two scenarios cannot be *distinguished* by processor p if p is running the same algorithm in both scenarios, and at all physical clock times T, it has the same message history in both scenarios. From our definitions it follows that if p is running a deterministic algorithm in two senarios that it cannot distinguish, then it performs the same actions in both scenarios at all physical clock times T. This observation will be crucial in the proof of our impossibility results.

3. Synchronization without Authentication

Consider the following definition of clock synchronization:

Weak Clock Synchronization Condition (*WCSC*): There exist constants PER, DMAX, and ADJ such that, for correct processors, (a) TAR is constant except that it changes at logical clock times that are positive multiples of PER by an amount with absolute values less than ADJ, and (b) the difference between logical clocks is always bounded by DMAX.

As we mentioned in the Introduction, if we do not require that $C(t)$ be unbounded, then there is a trivial algorithm to achieve WCSC. We choose PER arbitrarily. Then whenever $C(t) = \text{PER}$, we set $\text{TAR}(t^+) = -D(t)$ (thus making $C(t^+) = 0$). Clearly we have $\text{ADJ} = \text{DMAX} = \text{PER}$ in this case, since $C(t)$ is always between 0 and PER for a correct processor. However, even if we require that $C(t)$ have unbounded range, we still get:

THEOREM 1. *There is an algorithm that achieves* WCSC, *independent of the number of faults, for which $C(t)$ is unbounded.*

Proof. For ease of exposition, we assume that all correct processors start simultaneously at time 0, so that $D_p(0) = 0$ for a correct processor p. The general case (where processors start within some constant B of each other) works essentially the same way, and is left to the reader.

The idea of the algorithm is to have a processor's logical clock running at roughly the logarithm (base 2) of its physical clock. We proceed as follows.

Choose $\text{PER} = 2$ units. (This choice is made simply to make some calculations easier, since our logarithms are to the base 2. Actually, any choice of PER would work.) For each time t such that $C(t) = 2i$ $(= i\,\text{PER})$ for some positive integer i, and $0 \leqslant \log(D(t)) \leqslant 2i - 2$, we set $\text{TAR}(t^+) = \log(D(t)) - D(t)$, thus making $C(t^+) = \log(D(t))$. (Note that the purpose of checking that $\log(D(t)) \leqslant 2i - 2$, or equivalently, that $C(t) - \log(D(t)) \geqslant 2$, is simply to prevent TAR from being reset infinitely often in a bounded amount of time. Without this clause, for example, we would have $C(t) = 2$ for infinitely many values of t.)

We now show that if p is a correct processor following this algorithm, then, as illustrated in Fig. 1, we have

$$\max(0, \log(D_p(t))) \leqslant C_p(t) \leqslant \max(4, 4 + \log(D_p(t))). \qquad (*)$$

Suppose we can prove $(*)$. Since, by assumption, for a correct processor p we have $(1/R)t \leqslant D_p(t) \leqslant Rt$, it follows that $-\log(R) + \log(t) \leqslant \log(D_p(t)) \leqslant$

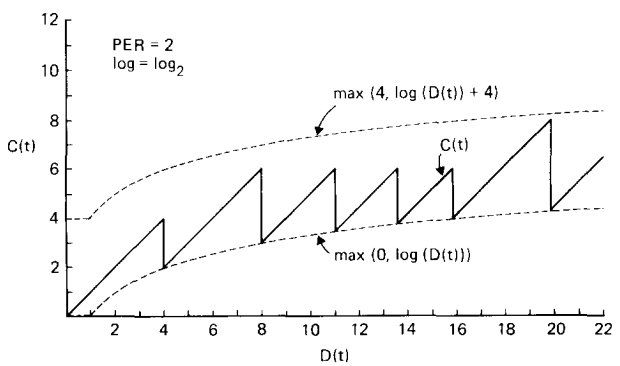

FIG. 1. A logarithmic envelope allows synchronization without messages.

$\log(R)+\log(t)$. Thus, from (*) it immediately follows that if p and q are correct processors, then $C_p(t)$ is unbounded and $|C_p(t)-C_q(t)| \leqslant 4+2\log(R)$. Moreover, note that every time TAR is changed, it is changed by an amount $C_p(t)-\log(D_p(t))$. Since $\log(D_p(t)) \geqslant 0$ when TAR is changed, it follows from (*) that TAR is changed by at most 4. Thus the algorithm achieves WCSC with PER $=2$, DMAX $=4+2\log(R)$, and ADJ $=4$.

So now it only remains to prove (*). The lower bound is immediate from the description of the algorithm, since $C_p(t)$ is set back to $\log(D_p(t))$ and then increases along with $D_p(t)$ until it is next reset; the algorithm also guarantees that $C_p(t)$ is always positive. For the upper bound, we will show that for all real times t such that $D_p(t) \geqslant 4$, we have $C_p(t) \leqslant 4+\log(D_p(t))$; since clearly $C_p(t) \leqslant D_p(t)$, this will complete the proof of (*).

Note that $f_p(t)=C_p(t)-\log(D_p(t))$ is continuous and increasing in the intervals in which TAR is constant. Moreover, $f_p(t^+)=0$ at those times t when TAR is changed. Suppose, to obtain a contradiction, that for some time t_1 such that $D_p(t_1) \geqslant 4$, we have $f_p(t_1)>4$. Then there must exist a real time $t_2<t_1$ such that $f_p(t_2)=2$, and TAR is not changed in the interval between t_2 and t_1. Since C_p is continuous in this interval, there must be a latest real time t' with $t_2 \leqslant t'<t_1$ such that $C_p(t')=2k$ for some integer $k>0$. Since $f_p(t') \geqslant 2$, we must have $\log(D_p(t')) \leqslant 2k-2$. And since $t' \geqslant 2$, we must have $\log(D_p(t')) \geqslant 0$, and, according to our algorithm, TAR would be changed at real time t', contradicting the initial assumptions. ∎

Note that this algorithm achieves WCSC *using no message exchanges* (although, as noted in the previous section, we may need some message exchanges to ensure that at the first time t_0 when all the processors are up, we have $D_p(t_0) \leqslant B$ for some constant B depending only the communication network).

In our algorithm, we essentially kept $C(t)$ to within a constant of $\log(D(t))$. It is easy to see that we could also have achieved WCSC by keeping $C(t)$ to within a constant of any linear function of $\log(D(t))$. To achieve an impossibility result, the requirements for clock synchronization must be somewhat strengthened. This is done by requiring that $C(t)$ stay within a linear function of $D(t)$. Before we formalize this notion of Linear Envelope Synchronization, we give another condition that implies it, which is more in the spirit of the clock synchronization condition of [LM].

Note that in the algorithm given in Theorem 1, for fixed i, there may be several times t when $C(t)=i$ PER and TAR is reset. If, for each i, we only allow changes in TAR the first time that $C(t)$ reads i PER, then the time between changes in this algorithm can grow unboundedly large; thus there is no bound on the difference between the logical clocks of correct processors. This leads us to make the following definition:

Clock Synchronization Condition (*CSC*): There exist constants PER, DMAX, and ADJ with ADJ < PER such that, for all correct processors, (a) TAR is constant except that it changes at logical clock times that are positive multiples of PER by

an amount with absolute value less than ADJ, (b) these changes can occur only the first time C reads i PER (i.e., if a change occurs when $C(t) = i$ PER, then $C(t') \neq i$ PER for all $t' < t$), and (c) the difference between logical clocks is always bounded by DMAX.

We need to introduce one more general notion of synchronization in order to get a precise statement of our results:

(U, L) *Envelope Synchronization.* Given two real-valued functions $U(t)$ and $L(t)$, (U, L) envelope synchronization is achieved if the logical clock C of a correct processor with duration timer D is bounded above by $U(D(t))$ and bounded below by $L(D(t))$ (i.e., $L(D(t)) < C(t) < U(D(t))$), and there is a constant DMAX such that logical clocks of correct processors differ by at most DMAX. A special case of (U, L) envelope synchronization is *linear envelope synchronization*, which we abbreviate as LES, where L and U are taken to be the linear functions $at + b$ and $ct + d$, respectively, with $a > 0$.

Linear envelope synchronization guarantees that the logical clock time of a correct processor is within a linear envelope of the physical clock time. But since we have assumed that physical clock time is within a linear envelope of real time (bounded by R and $1/R$), LES also implies that for a correct processor, logical clock time is within a linear envelope of real time.

PROPOSITION 1. *An algorithm that achieves* CSC *achieves* LES.

Proof. Suppose $\mathscr{A}$ is an algorithm that achieves CSC with parameters PER and ADJ. For a correct processor p, TAR can be changed only when $C_p(t)$ is a multiple of PER. If p sets its clock forward by the maximum allowable amount ADJ at every opportunity, $C_p(t)$ will read the next multiple of PER every time PER-ADJ passes on $D_p(t)$. If p sets its clock backward by ADJ at every opportunity, $C_p(t)$ will read the next multiple of PER every time PER + ADJ has passed on $D_p(t)$. To see this, note that if TAR is set back by ADJ when $C_p(t) = k$ PER, then $C_p(t^+) = k$ PER-ADJ. This means that $C_p(t^+) \geq (k-1)$PER, since by assumption ADJ < PER. Thus the next allowable time to change TAR is when $C_p(t) = (k+1)$ PER, after PER + ADJ has passed on $D_p(t)$, since TAR can be changed only the first time $C_p(t)$ reads k PER. Since it is possible that $C_p(0) =$ PER (so that the first change can be made at time 0), it can be shown that we must have, for every correct processor p, $(\text{PER}/(\text{PER} + \text{ADJ}))(D_p(t) - D_p(0)) - \text{ADJ} \leqslant C_p(t) - C_p(0) \leqslant (\text{PER}/(\text{PER-ADJ}))(D_p(t) - D_p(0)) + \text{ADJ}$. It now easily follows that $\mathscr{A}$ achieves LES. ∎

The proof of Theorem 1 above shows that $(t, \log(t))$ envelope synchronization is achievable. Theorem 2 below states that LES is not achievable in a network with uncertain transmission times if at least one-third of the processors are faulty. And thus by Proposition 1, CSC is also not achievable if at least one-third of the processors are faulty.

THEOREM 2. *Linear envelope synchronization is impossible in a network with uncertain transmission times when one third or more of the processors are faulty.*

Before we can prove Theorem 2, we need to establish some notation. Suppose N is communication network with uncertain transmission times consisting of n processors. Let $\rho = \min(R, \{H(p, q)/L(p, q) \mid (p, q)$ is a link in N, $L(p, q) \neq 0\})$. Choose $r_1, \ldots, r_n$ with $1 \leqslant r_i \leqslant \rho$, $i = 1, \ldots, n$. We define a *standard scenario* determined by $r_1, \ldots, r_n$ to be one where $D_i(t) = r_i t$ and, if there is a link in N from p_i to p_j, all messages from p_i to p_j along that link have transmission time $(\rho/r_j)\, L(p_i, p_j)$. (So, in particular, if $L(p_i, p_j) = 0$, then the transmission time from p_i to p_j is 0.) Since $1 \leqslant \rho/r_j \leqslant \rho$, and $\rho L(p_i, p_j) \leqslant H(p_i, p_j)$, this is a legal transmission along this link. Note that for all choices of algorithms $\mathscr{A}_1, \ldots, \mathscr{A}_n$ and all choices of $r_1, \ldots, r_n$ with $1 \leqslant r_i \leqslant \rho$, there is a standard scenario determined by $r_1, \ldots, r_n$ where p_i runs $\mathscr{A}_i$. For the remainder of this section, we consider only standard scenarios.

We say that an algorithm $\mathscr{A}$ *achieves* LES *in network N with parameters* DMAX, a $(a > 0)$, b, c, *and* d, *and tolerates* f *faults* if whenever all the correct processors in N run $\mathscr{A}$ and there are at most f faults, for all correct processors p_i and p_j we have

$$|C_i(t) - C_j(t)| \leqslant \text{DMAX},$$

$$aD_i(t) + b \leqslant C_i(t) \leqslant cD_i(t) + d.$$

We need one more definition before we can state the crucial lemma on which the proof of Theorem 2 depends. If $\mathscr{B}$ and $\mathscr{C}$ are algorithms, let $\mathscr{B} + \mathscr{C}[q]$ be the algorithm which has the following properties: processor p, running this algorithm, sends the same messages to all processors other than q as it would if it were running $\mathscr{B}$, and it sends the same messages to q as it would if it were running $\mathscr{C}$. However, we must be a little careful here when we say "as if it were running $\mathscr{B}$" or $\mathscr{C}$, since p's message history when running $\mathscr{B} + \mathscr{C}[q]$ will include the messages it sent according to both $\mathscr{B}$ and $\mathscr{C}$. Thus, when computing what to do according to algorithm $\mathscr{B}$, all the messages sent to q according to $\mathscr{C}$ should be deleted. Similar remarks hold when computing what messages to send according to $\mathscr{C}$. Thus, by running $\mathscr{B} + \mathscr{C}[q]$, processor p displays "two-faced" behavior, pretending to processor q that it is running $\mathscr{C}$ and pretending to all other processors that it is running $\mathscr{B}$. Processor p *sends messages to* q *according to algorithm* $\mathscr{C}$ if it is running $\mathscr{B} + \mathscr{C}[q]$ for some algorithm $\mathscr{B}$.

The following lemma provides the key step of our proof of Theorem 2 in the case of three processors, one of which is faulty. Roughly speaking, it says that, for all n, there is an algorithm that the faulty processor can run which forces the logical clock of one of the correct processors to run at a rate greater than $a\rho^n$, no matter what messages the other correct processor sends. For ease of exposition, we have done the three-processor case first: we consider the general case in Lemma 2 below.

LEMMA 1. *Let N be a communication network with uncertain transmission times consisting of three processors, and let $\mathscr{A}$ be an algorithm which achieves* LES *in N*

with parameters DMAX, *a, b, c, d and tolerates one fault. For all integers* $n > 0$, *all permutations ijk of* $\{1, 2, 3\}$, *and all* r_i, r_j, r_k *with* $1 \leq r_i, r_j, r_k \leq \rho$, *there is an algorithm* $\mathscr{F}_n(ijk, r_i, r_j, r_k)$ *such that in the standard scenario with* $D_h(t) = r_h t$, $h = 1, 2, 3$, *where* p_i *sends messages to* p_j *according to* $\mathscr{F}_n(ijk, r_i, r_j, r_k)$, *and* p_j *and* p_k *are correct and run* $\mathscr{A}$, *we have, for all real times t,* $C_j(t) > a\rho^n D_j(t) + b - n$ DMAX.

Proof. Fix N and $\mathscr{A}$ as in the hypothesis of the lemma. We define $\mathscr{F}_n(ijk, r_i, r_j, r_k)$ for all choices of r_i, r_j, r_k, by induction on n. For the case $n = 0$, for all choices of r_i, r_j, r_k, we take $\mathscr{F}_0(ijk, r_i, r_j, r_k) = \mathscr{A}$. Since $\mathscr{A}$ achieves LES with parameters DMAX, a, b, c, d, and tolerates one fault, then as long as p_j and p_k both run $\mathscr{A}$, $C_j(t) > aD_j(t) + b$ must hold in *all* scenarios, no matter what algorithm p_i runs.

Suppose we have proved the result for $n = m$. We first give an informal proof of the result for $n = m + 1$. When running $\mathscr{F}_{m+1}$ (we omit the parameters to $\mathscr{F}$ in this informal description). p_i pretends that it is running $\mathscr{A}$, with its physical clock running at a rate ρ times the rate of p_j, and that p_k is sending it messages that result in $C_i(t) > a\rho^m D_i(t) + b - m$ DMAX (by running $\mathscr{F}_m$). Now p_j cannot distinguish a scenario where p_i is running $\mathscr{F}_{m+1}$ and p_j and p_k are both running $\mathscr{A}$ from one where p_i is correctly running $\mathscr{A}$ with its physical clock at a rate ρ times that of p_j, p_j is running $\mathscr{A}$, and p_k is sending p_i messages according to $\mathscr{F}_m$. In the latter scenario, by inductive hypothesis and the fact that $\mathscr{A}$ achieves LES and tolerates one fault, we have

$$
\begin{aligned}
C_j(t) &> C_i(t) - \text{DMAX} \\
&> a\rho^m D_i(t) + b - (m+1)\,\text{DMAX} && \text{(by inductive hypothesis)} \\
&= a\rho^{m+1} D_j(t) + b - (m+1)\,\text{DMAX} && \text{(since } D_i(t) = \rho D_j(t)\text{)}.
\end{aligned}
$$

Since p_j cannot distinguish these two scenarios, this inequality must also hold in the former scenario.

The problem with our informal proof above is that it does not suffice to say, for example, "p_i pretends ... that p_k is ... running $\mathscr{F}_m$." We must also specify the rest of the scenario, and in particular, what p_k's message history is when it runs $\mathscr{F}_m$. In order to formalize these notions, in particular the idea of constructing an algorithm where a processor "pretends" that another processor is trying to "fool" it, we need two operators on message histories and physical clocks.

(1) If Σ is a scenario and β is a real-valued function, let $mimic(\beta, \Sigma)$ be the algorithm that, when applied by processor p to a pair (T, H) consisting of a physical clock reading T and a message history H, sends message m to q if and only if $\langle \beta(T), m, p, \text{"received"} \rangle$ is in the message history of q in Σ. If $\mathscr{S}$ is a mapping which takes a message history and returns a scenario, let mimic*$(\beta, \mathscr{S})$ be the mapping that returns mimic$(\beta, \mathscr{S}(H))(T, H)$ when applied to a pair (T, H). Note that for mimic* to compute what message to send q at physical clock time T according

to algorithm mimic(β, Σ), processor p must simulate Σ until the simulated physical clock of processor q reads $\beta(T)$.

It is easy to see that mimic(β, Σ) has the following crucial property: Suppose Σ' is a scenario such that any message sent by p at time T on p's physical clock in Σ' will be received by q at time $\beta(T)$ on q's physical clock. Then if p runs mimic(β, Σ) in Σ', q receives the same messages from p at the same physical clock times in both Σ and Σ'. We will be making frequent use of this property throughout our proof.

(2) Let *output* be a mapping from message histories to algorithms such that, if H is a message history, then output(H) is an algorithm that, when applied to a pair (T, H') by processor p, sends message m to processor q at time T on its physical clock if and only if $\langle T, m, q, \text{"sent"} \rangle$ is in H. Thus, when p runs output(H), it sends exactly the same messages to all processors as in message history H, regardless of what messages it gets from the other processors. Note that this means that a processor running output(H) will stop sending messages after the greatest timestamp on any of the messages sent in history H.

We now construct $\mathscr{F}_{m+1}(ijk, r_i, r_j, r_k)$ so that for each algorithm $\mathscr{B}$, there exists an algorithm $\mathscr{D}$ such that the standard scenarios Σ and Φ described in the diagram below cannot be distinguished by p_j. These scenarios correspond to the two scenarios in our informal description above. Note that in scenario Φ, p_i's physical clock is running at ρ times the rate of p_j's and that p_k is sending p_i messages according to $\mathscr{F}_m(kij, 1, \rho, 1)$.

Scenario Σ

Processor	i	j	k
Algorithm	$\mathscr{B} + \mathscr{F}_{m+1}(ijk, r_i, r_j, r_k)[j]$	$\mathscr{A}$	$\mathscr{A}$
Physical clock	$r_i t$	$r_j t$	$r_k t$

Scenario Φ

Processor	i	j	k
Algorithm	$\mathscr{A}$	$\mathscr{A}$	$\mathscr{D} + \mathscr{F}_m(kij, 1, \rho, 1)[i]$
Physical clock	ρt	t	t

In Φ by inductive hypothesis we have $C_i(t) > a\rho^m D_i(t) + b - m\,\text{DMAX}$. The inequalities in the informal argument above show that in Φ we must have $C_j(t) > a\rho^{m+1} D_j(t) + b - (m+1)\,\text{DMAX}$, so this inequality also holds for Σ, as desired, since these two scenarios cannot be distinguished by p_j.

Thus it suffices to find $\mathscr{F}_{m+1}(ijk, r_i, r_j, r_k)$ with the desired property. In order to give a precise definition of $\mathscr{F}_{m+1}(ijk, r_i, r_j, r_k)$, we first need to define two auxiliary functions, Φ^* and Σ^*, from message histories to standard scenarios. (The numbers r_i, r_j, and r_k that appear as parameters to $\mathscr{F}_{m+1}$ will also be parameters of Φ^* and Σ^*, but we suppress them here.) Given a message history H (which in the proof will be taken to be p_i's message history at various times in scenario Σ), $\Sigma^*(H)$ and $\Phi^*(H)$ are defined by the diagram

Scenario $\Sigma^*(H)$

Processor	i	j	k
Algorithm	output(H)	$\mathscr{A}$	$\mathscr{A}$
Physical clock	$r_i t$	$r_j t$	$r_k t$

Scenario $\Phi^*(H)$

Processor	i	j	k
Algorithm	$\mathscr{A}$	$\mathscr{A}$	mimic$(\beta, \Sigma^*(H)) + \mathscr{F}_m(kij, 1, \rho, 1)[i]$
Physical clock	ρt	t	t

where $\beta(T) = T + \rho L(p_k, p_j)$.

Now let $\mathscr{F}_{m+1}(ijk, r_i, r_j, r_k) = \text{mimic}^*(\alpha, \Phi^*)$, where $\alpha(T) = (r_j/r_i)T + \rho L(p_i, p_j)$, and let the algorithm $\mathscr{D}$ of scenario Φ be defined via $\mathscr{D} = \text{mimic}(\beta, \Sigma)$. We now show that Σ and Φ cannot be distinguished by p_j. Since a message sent by p_k to p_j (if there is a link between them) in scenario Φ at time T on p_k's physical clock is, by the definition of transmission times in standard scenarios, received by p_j at time $\beta(T)$ on its physical clock, the crucial property of mimic guarantees that p_j receives the same messages from p_k at the same physical clock times in both Σ and Φ.

Thus, in order to show that Σ and Φ cannot be distinguished by p_j, it only remains to show that p_j gets the same messages at the same physical clock times from p_i in both scenarios. Let $H_i(t)$ be p_i's message history in scenario Σ at real time t. It is easy to see that, by construction, scenarios Σ and $\Sigma^*(H_i(t))$ are identical up to real time t. Using this fact we can also show that scenarios Φ and $\Phi^*(H_i(t))$ are identical up to real time t (since everything is identical in the two scenarios except than in Φ, p_k sends messages to p_j according to mimic(β, Σ), while in $\Phi^*(H_i(t))$, p_k sends messages to p_j according to mimic$(\beta, \Sigma^*(H_i(t)))$). Finally, we claim that p_j receives the same messages at the same physical clock time from p_i in scenarios $\Phi^*(H_i(t))$ and Σ up to real time t. Note that once we prove this claim, it follows immediately by transitivity that p_j receives the same messages at the same physical clock times in scenarios Φ and Σ.

To prove the claim, suppose that p_j receives a message m from p_i at real time $t' < t$ in scenario Σ, and this at physical clock time $r_j t'$. Message m must have been sent by p_i at real time $t'' = t' - (\rho/r_j) L(p_i, p_j)$, and thus at physical clock time $T'' = r_i t''$. Since p_i sends messages to p_j in scenario Σ according to mimic$^*(\alpha, \Phi^*)$, this means that p_j receives m at $\alpha(T'') = r_j t'$ in scenario $\Phi^*(H_i(t''))$. Now to complete the proof we must still show that p_j also receives m at physical clock time $r_j t'$ in scenario $\Phi^*(H_i(t))$. We will prove this by proving a more general fact: namely,

> if $u \leqslant u'$, then p_i sends the same messages in $\Phi^*(H_i(u))$ and $H^*(H_i(u'))$ up to real time $r_j u$. (†)

From (†) it follows that p_i sends the same messages in $\Phi^*(H_i(t''))$ and $\Phi^*(H_i(t))$ up to real time $r_j t''$. Now p_j receives m at physical clock time (and hence also real time) $r_j t'$ in $\Phi^*(H_i(t''))$; thus p_i sent m in $\Phi^*(H_i(t''))$ at real time $r_j t' - \rho L(p_i, p_j) = r_j t''$. From (†) we get that p_i also sent m at real time $r_j t''$ in scenario $\Phi^*(H_i(t))$, and

hence that p_j received m at real time (and physical clock time) $r_j t'$ in $\Phi^*(H_i(t))$. Thus it only remains to prove (†).

Clearly $\Phi^*(H_i(u))$ and $\Phi^*(H_i(u'))$ are identical except that p_k sends p_j messages according to mimic$(\beta, \Sigma^*(H_i(u)))$ in one case, and mimic$(\beta, \Sigma^*(H_i(u')))$ in the other. From the definition of mimic, it follows that p_k sends the same messages to p_j in both scenarios up to real tume $r_j u - \rho L(p_k, p_j)$. We leave it to the reader to check that this means that all processors will receive the same messages in both scenarios up to real time $r_j u$. Since p_i is running the same algorithm in both $\Phi^*(H_i(u))$ and $\Phi^*(H_i(u'))$, it must send the same messages in both scenarios up to real time $r_j u$. This completes the proof of (†) and of the lemma. ▮

We now generalize Lemma 1 to arbitrary networks. The generalization intuitively says that if one-third or more of the processors in the network are faulty, then for all n, there is an algorithm that the faulty processors can run which forces the logical clocks of a non-empty subset of correct processors to run at a rate greater than $a\rho^n$, no matter what messages the other correct processors send.

LEMMA 2. *Let N be a communication network with uncertain transmission times consisting of m processors, and let $\mathscr{A}$ be an algorithm which achieves* LES *in N with parameters* DMAX, *a, b, c, d and tolerates f faults, where $3f \geqslant m$. For all integers $n > 0$, all partitions P_1, P_2, P_3 of the processors in N into three disjoint sets, with $1 \leqslant |P_h| \leqslant f$, $h = 1, 2, 3$, all permutations ijk of $\{1,2,3\}$, and all r_i, r_j, r_k with $1 \leqslant r_i, r_j, r_k \leqslant \rho$, there is an algorithm $\mathscr{F}_n(ijk, r_i, r_j, r_k)$ such that in the standard scenario with $D_p(t) = r_h t$ for all processors $p \in P_h$, $h = 1, 2, 3$, where all processors in P_i send messages to the processors in P_j according to $\mathscr{F}_n(ijk, r_i, r_j, r_k)$, and the processors in P_j and P_k are correct and run $\mathscr{A}$, we have, for all real times t, $C_p(t) > a\rho^n D_p(t) + b - n$* DMAX *for all processors p in P_j.*

Proof. The proof is a straightforward generalization of that of Lemma 1. We construct scenarios Σ and Φ just as above. The only added point that must be checked is that all the processors in P_j get the same messages from the other processors in P_j in both Σ and Φ. The processors in P_j are following $\mathscr{A}$ in both scenarios, but in scenario Σ, their physical clocks are running at r_j times the rate of real time, while in Φ they are running at exactly the rate of real time. However, because we are considering standard scenarios, messages to processors in P_j take r_j times as much real time to arrive in scenario Φ as in scenario Σ. Thus, in both scenarios, the message transmission time as measured on the physical clocks of processors in P_j is the same. The result easily follows. ▮

Proof of Theorem 2. Suppose by way of contradiction that N is a network with uncertain transmission times consisting of m processors, and $\mathscr{A}$ is an algorithm that achieves LES in N with parameters DMAX, a, b, c, and d, and tolerates f faults, with $3f \geqslant m$. Let P_1, P_2, P_3 be a partition of N such that $1 \leqslant |P_h| \leqslant f$, for $h = 1, 2, 3$, and choose n such that $a\rho^n > c$. Consider a standard scenario where all physical clocks are running at the rate of real time, all the processors in P_1 are faulty and

running algorithm $\mathscr{F}_n(123, 1, 1, 1)$, while the processors in P_2 and P_3 are all correct and running $\mathscr{A}$. By Lemma 2, we have $C_p(t) > a\rho^n D_p(t) + b - n$ DMAX, for all processors $p \in P_2$ and all times t. Since $a\rho^n > c$ and $D_p(t) = t$, there must be some t' such that $a\rho^n D_p(t) + b - n$ DMAX $> cD_p(t) + d$ for all $t > t'$. This contradicts the assumption that $\mathscr{A}$ achieves LES in N with parameters DMAX, a, b, c, and d, and tolerates f faults. ∎

We have just shown that even in a completely connected network, if one-third or more of the processors are faulty, LES is not achievable. But in general, networks are not completely connected. Dolev [Do] has shown that Byzantine agreement is not achievable if the connectivity of the network is less than $2f + 1$ and there are f faulty processors. By combining the techniques of the proof of Theorem 2 with some of the techniques of [Do], we can also show that linear envelope clock synchronization is impossible if the connectivity of the network is less than $2f + 1$ and there are f or more faulty processors. A proof of this fact (as well as another elegant proof of Theorem 2) can be found in [FLM].

The reader may wonder if we really need our assumption that the network has uncertain transmission times. That is, would our results still hold in a network N where $H(p, q) = L(p, q)$ for all links (p, q), so that there is no uncertainty in transmission times? It can be shown that Theorem 2 still hold in a network where $H(p, q) = L(p, q) = 0$ (essentially the same proof goes through), but, as shown by Fischer, Lynch, and Merritt [Me], the result does not hold if $H(p, q) = L(p, q) > 0$.

THEOREM 3 (Fischer, Lynch, Merritt). *Let N be a network with uncertainty in transmission times such that $H(p, q) = L(p, q) > 0$ for all links (p, q). Then there is an algorithm $\mathscr{A}$ that achieves linear envelope synchronization and tolerates f faults iff N has indegree $2f + 1$ (i.e., each processor can receive directly from at least $2f + 1$ other processors).*

Proof sketch. For the upper bound, suppose N is a network with no uncertainty and indegree $2f + 1$. Assume for ease of exposition that $C_p(0) = 0$ for all correct processors p; the general case is similar and left to the reader. We show that there exists a constant DMAX such that if p is a correct processor, we can keep C_p within DMAX of real time, i.e., $|C_p(t) - t| <$ DMAX. Thus clearly we can achieve LES.

The basic idea of the algorithm is that in a network with no uncertainty, a processor can know exactly how much real time has passed by measuring the time required to send a message to its neighbor and receive a response. We present the algorithm from the point of view of processor p. Initially p sends an INIT(q) message to each neighbor q. Whenever p receives a message from a neighbor, it immediately returns the message. Note that in particular, the INIT(q) message will keep being passed and forth from p to q. We will call a subset S of processors of size $f + 1$ consisting of neighbors of p *good at real time t* if the order in which INIT(q) messages are received by $q \in S$ is exactly what it would be if all the processors in S were correct. Processor p can easily check if S is good at time t,

since it known what the message transmission time should be for each message. Since there is a subset of size $f+1$ which consists of only correct processors, there must be at least one good subset according to p at all times t. Moreover, any good subset contains at least one correct processor (since there are at most f faulty processors).

Assume that p has ordered all the subsets of its neighbors of size $f+1$ in some way, and, for each such subset S, it has chosen some processor $q_S \in S$ to "trust." For each subset S, p constructs a logical clock $C_{p,S}$ by initializing $C_{p,S}$ to 0 and then setting $C_{p,S}$ to $k\delta(p, q_S)$ when it receives the kth INIT(q_S) message from q_S, where $\delta(p, q)$ is the real time required for a message to make the round trip from p to q and then back from q to p. We take $C_p(t)$ to be $C_{p,S}(t)$ where S is the first subset in the order that is good at time t.

We claim that $|C_p(t) - t| \leqslant 2\Delta$, where Δ is the maximum round trip message time from p to any of its neighbors. To prove the claim, we now show that if S is good at time t, then $|C_{p,S}(t) - t| \leqslant 2\Delta$. Since S is good at time t, there must be at least one correct processor in S, say q'. Of course, the processor q_S in S that p has chosen to "trust" may be faulty. Suppose that p has received k INIT(q_S) messages from q_S by real time t, and thus has set $C_{p,S}$ to $k\delta(p, q_S)$. Further suppose that p has received k' INIT(q') messages from q' by time t. Since q' is correct, this means that

$$(1) \quad k'\delta(p, q') \leqslant t \leqslant (k'+1)\,\delta(p, q').$$

And we must have

$$(2) \quad (k+1)\,\delta(p, q_S) \geqslant k'\,\delta'(p, q').$$

since otherwise the $(k+1)$st INIT(q_S) message would have been received before the (k')th INIT(q') message, contradicting the fact that S is good at time t. From (1) we have that $k'\,\delta(p, q') \geqslant t - \delta(p, q')$, which together with (2) gives

$$(3) \quad C_{p,S}(t) = k\,\delta(p, q_S) \geqslant t - \delta(p, q_S) - \delta(p, q').$$

We must also have

$$(4) \quad k\,\delta(p, q_S) \leqslant (k'+1)\,\delta(p, q'),$$

since otherwise the kth INIT(q_S) message would have been received after the $(k'+1)$st INIT(q') message, again contradicting the fact that S is good at time t. From this and (1) it follows that

$$(5) \quad C_{p,S}(t) \leqslant k'\,\delta(p, q') + \delta(p, q') \leqslant t + \delta(p, q').$$

Since, by choice, $\Delta \geqslant \max\{\delta(p, q_S), \delta(p, q')\}$, from (3) and (5) it follows that $|C_{p,S}(t) - t| \leqslant 2\Delta$, as desired.

The lower bound can be proved using techniques similar to those of Theorem 2, or by using the techniques developed in [FLM]. We leave details to the reader. ∎

It is interesting to note that the upper bound of Theorem 3 can be achieved even if the correct processors do not have physical clocks. (This follows immediately

from the observation that nowhere in the algorithm did we need to assume the existence of physical clocks.)

4. Lower Bounds on Synchronization

Suppose we have an algorithm that guarantees that logical clocks of correct processors are no more than DMAX apart at any real time. It is easy to see that, for all $\varepsilon > 0$, we can modify this algorithm to obtain an algorithm that guarantees that logical clocks of correct processors are no more than ε DMAX apart at all times, simply by slowing down all clocks by a factor of ε. Of course, the slope of linear envelope limits the choice of ε.

These observations suggest that if we want to get lower bounds on how tightly logical clocks can be synchronized, it may not be appropriate to consider the difference between logical clock readings at a given real time. We therefore turn our attention from the tightness of synchronization along the logical clock time axis to the tightness of synchronization along the real time axis. We show that there is a lower bound Δ, which depends on the uncertainty of transmission delay, such that no clock synchronization algorithm can guarantee that the difference between the real times at which clocks read a given value is less than Δ. In fact, we prove an even stronger result: we show that there is no algorithm that can guarantee that *any* action can be performed by two processors within less than Δ of each other, for an appropriately defined notion of action. These results thus give lower bounds on the degree of synchronization achievable in a network. We call Δ the *essential temporal imprecision*, or just *imprecision*, of the network.

To make these notions precise, consider a communication network $N = (G, H, L)$. For the purposes of this section, we will assume that G is an undirected graph (so that there is a link (p, q) iff there is a link (q, p)) and that $H(p, q) = H(q, p)$ and $L(p, q) = L(q, p)$. (The case of directed networks is considered in [HMM]. It is shown in [HMM] that there is a precise sense in which for calculating imprecision, there is no loss of generality in considering only undirected networks where H and L are symmetric.) If p and q are joined by a link, define $V(p, q)$, the *variation* in uncertainty between p and q, via $V(p, q) = H(p, q) - L(p, q)$. We extend H, L, and V so that they apply to all pairs of processors by setting $H(p, q) = L(p, q) = V(p, q) = \infty$ for processors p, q, that are not connected by a direct link. We now extend V so that it also applies to sequences of processors. For a sequence of processors $\pi = p_0, p_1, \ldots, p_n$, let $V(\pi)$ be the sum of the values $V(p_i, p_{i+1})$, for i from 0 to $n-1$. Finally, let $U_N(p, q)$, the *uncertainty* in transmission time from p to q in N, be defined via $U_N(p, q) = \min\{V(\pi) \mid \pi$ is a sequence of processors in N starting with p and ending with $q\}$, and let $U_N = \max\{U_N(p, q) \mid p, q$ are processors in $N\}$.

For ease of exposition in what follows, we assume that each processor has a special register that initially contains the value 0. At some point the value must be changed to 1. The problem is to obtain an algorithm that guarantees that all

processors change the value to 1 at as close to the same real time as possible. Clearly the time at which a processor changes the value in its special register depends on the scenario. The essential temporal imprecision inherent in a particular algorithm $\mathscr{A}$ is the worst case difference in the times that two processors change the value in their special register, where the difference is taken over all possible scenarios in which *all* processors run algorithm $\mathscr{A}$. Note that since we are assuming that all processors run $\mathscr{A}$, we are implicitly ignoring the possibility of faults. Clearly any lower bounds we obtain under the assumption that there are no faults also holds if there are faults. Recall that a *run* of algorithm $\mathscr{A}$ in network N is a particular scenario in which all processors in N run algorithm $\mathscr{A}$.

The essential temporal imprecision in a network is the minimum essential temporal imprecision over all possible algorithms. More formally, given an algorithm $\mathscr{A}$, processors p and q in N, and run σ define

$$\begin{aligned}
\Delta_{N,\mathscr{A}}(p, q, \sigma) &= \text{the absolute value of the difference of the real times at which processors } p \text{ and } q \text{ change the value of their special register in run } \sigma \text{ of algorithm } \mathscr{A},\\
\Delta_{N,\mathscr{A}}(p, q) &= \max_{\sigma}\{\Delta_{N,\mathscr{A}}(p, q, \sigma)\},\\
\Delta_{N}(p, q) &= \min_{\mathscr{A}}\{\Delta_{N,\mathscr{A}}(p, q)\},\\
\Delta_{N,\mathscr{A}} &= \max_{p,q}\{\Delta_{N,\mathscr{A}}(p, q)\},\\
\Delta_{N} &= \min_{\mathscr{A}}\{\Delta_{N,\mathscr{A}}\}.
\end{aligned}$$

Thus $\Delta_{N,\mathscr{A}}(p, q)$ is the closest real time synchronization that can be *guaranteed* between p and q when running algorithm $\mathscr{A}$, $\Delta_N(p, q)$ is the closest real time synchronization that can be guaranteed between p and q, no matter what algorithm is run, $\Delta_{N,\mathscr{A}}$ is the imprecision in the network (the worst case real time difference between when two processors in N perform a given action) when running $\mathscr{A}$, and Δ_N is the imprecision in the network, the tightest coordination in N that can be guaranteed by *any* algorithm.

THEOREM 4. *For all communication networks N and all processors p, q in N, we have $\Delta_N(p, q) \geq U_N(p, q)/2$; i.e., the imprecision is at least as great as half the uncertainty.*

Proof. Fix a communication network $N=(G, H, L)$, an algorithm $\mathscr{A}$, and processors p and q in N. We consider two runs of $\mathscr{A}$ which, as we shall show, are indistinguishable from the point of view of every processor. In the first run, all processors are started at the same time, with their physical clocks running at the rate of real time. For definiteness, we take $D_r(t)=t$ for all processors r. If there is a link from processor r to processor r' in N, then messages from r to r' take time $L(r, r')+\max(U_N(p, r)-U_N(p, r'), 0)$. We leave it to the reader to check that this expression is at most $H(r, r')$, so a message from r to r' can indeed take this length of time.

In the second run, processor p starts first, each processor r starts at real time $U_N(p, r)$ later than p, and all processor's physical clocks run at the rate of real time. Again, for definiteness, we take $D_r(t) = t - U_N(p, r)$ for all processors r. If r and r' are joined by a link in N, then messages from processor r to r' take time $L(r, r') + \max(U_N(p, r') - U_N(p, r), 0)$ to arrive. Again it is easy to check that this is at most $H(r, r')$, and that no message reaches a processor before it has been started. We now show that these two runs are indistinguishable from the point of view of every processor; i.e., they produce the same message history.

Suppose r and r' are joined by a link in N, and r sends r' a message when r's physical clock reads T. First consider the case where $U_N(p, r) \geq U_N(p, r')$. In the first run, this message arrives at r' in time $L(r, r') + U_N(p, r) - U_N(p, r')$, when the physical clock of r' reads $T + L(r, r') + U_N(p, r) - U_N(p, r')$. In the second run, this message arrives at r' in time $L(r, r')$, but again the physical clock of r' reads $T + L(r, r') + U_N(p, r) - U_N(p, r')$, since r' is started $U_N(p, r) - U_N(p, r')$ ahead of r. The argument in the case where $U_N(p, r) < U_N(p, r')$ is similar and is omitted here.

Because messages are being sent and received at the same time on each processor's physical clock in both runs, processors perform the same action at a given time on their physical clocks in both runs. Suppose processor p changes the value of its special register at time T_1 on its physical clock, while processor q changes the value at time T_2 on its physical clock. Let t_1 and t_2 be the real times that the physical clocks of p and q read T_1 and T_2, respectively, in the first run. Note that in the second run, processor p's physical clock still reads T_1 at t_1, but processor q's physical clock reads T_2 at $t_2 + U_N(p, q)$ (since processor q was started $U_N(p, q)$ later in the second run). By definition, $\Delta_{N,\mathscr{A}}(p, q) \geq \max(|t_2 - t_1|, |t_2 + U_N(p, q) - t_1|)$. But it can be straightforwardly checked that this latter quantity is $\geq U_N(p, q)/2$. Since $\mathscr{A}$ was chosen arbitrarily, we get $\Delta_N(p, q) \geq U_N(p, q)/2$, as desired. ∎

Remarks. 1. The lower bound holds even if there are no faults in the network, and physical clocks run at exactly the rate of real time (i.e., there is no clock drift).

2. Even if processor's physical clocks are initially synchronized, we can also prove a version of this result in the presence of clock drift. We again consider two runs of $\mathscr{A}$ which we can show are indistinguishable from the point of view of any processor. The first run is defined just as in the proof of Theorem 4; in the second run, processors start with their physical clocks synchronized, but the physical clocks of processors drift until, for all r, p's physical clock is $U_N(p, r)$ ahead of that of r. From this point we can essentially repeat the proof above.

3. We can also extend this result to probabilistic algorithms (where processors can base their decisions on the results of coin tosses) in the following way. Consider the two runs described in the proof above, and suppose each of them holds with probability 1/2. Then an argument essentially identical to the one above shows that even for a probabilistic algorithm, the mean difference between the times

that processors p and q change the value of the special register in these two runs is at least $U_N/2$. (See [HMM] for further details and a generalization of this result.)

COROLLARY 1. *For all communication networks N, we have $\Delta_N \geq U_N/2$.*

Proof. Note that $\Delta_N \geq \max_{p,q} \Delta_N(p, q)$, since we cannot do worse by allowing different algorithms to synchronize different pairs of processors rather than using the same algorithm to synchronize all pairs. The result follows now immediately from Theorem 4 and the definitions. ∎

THEOREM 5. *For all $\varepsilon > 0$, there exists a communication network N such that $\Delta_N > U_N - \varepsilon$.*

Proof. Fix $\varepsilon > 0$. Choose $\eta > 0$ and such that $\eta/n < \varepsilon$. Let $N = (G, H, L)$, where G is a completely connected graph with n nodes, $H(p, q) = \eta$, and $L(p, q) = 0$, for all nodes p, q in G. Note we then have $U_N = \eta$. In [LL2], it is shown that $\Delta_N = ((n-1)/n)\eta$. Since $((n-1)/n)\eta > U_N - \varepsilon$, the desired result immediately follows. ∎

Theorem 5 is essentially the best we can do, as the following theorem shows.

THEOREM 6. *There exists a clock synchronization algorithm $\mathcal{A}$ such that for all communication networks N and processors p and q in N, $\Delta_{N,\mathcal{A}}(p, q) \leq U_N(p, q)$; i.e., the imprecision is no greater than the uncertainty.*

Proof. The clock synchronization algorithm of [HSSD] can be used to guarantee that for all T, there is a $T' > T$ such that the real times at which the logical clocks of processors p and q read T' differ by at most $U_N(p, q)$ in every run. (It follows from the algorithm of [HSSD] that the time T' is the local time when a resynchronization occurs, and is of the form k PER.) Each processor can thus use its logical clock to decide when to change the value in its special register. ∎

COROLLARY 2. *For all communication networks N, $\Delta_N \leq U_N$.*

We remark that the results of Theorems 4, 5, and 6, and their corollaries have recently been extended in [HMM], where an algorithm is given for computing the precise imprecision of any network. Since the algorithm of [HSSD] works even in the presence of faults (provided authentication is allowed) and clock drift, the upper bound of Theorem 6 also holds in this case. The interested reader may also wish to consult [HMM] for further results on achieving optimal precision in the presence of faults.

We present one final result considering tightness of synchronization for algorithms that achieve LES.

THEOREM 7. *If algorithm $\mathcal{A}$ achieves linear envelope synchronization in communication network N with parameters* DMAX, *a, b, c, and d, then* DMAX $\geq aRU_N/2$.

Proof. Suppose $\mathscr{A}$ is an algorithm that achieves LES with parameters DMAX, a, b, c, and d. Note that this implies that for any constant U and any processor p in the network, there must be a time t_0 such that $C_p(t_0+U)-C_p(t_0) \geqslant a(D_p(t_0+U)-D_p(t_0))$ (otherwise it is easy to see that we must have $C_p(t) < aD_p(t)+e$ for all times t and for some constant e, contradicting the fact that $\mathscr{A}$ achieves LES).

Consider two processors p, q in N such that $U_N(p, q) = U_N$. Now consider two runs of $\mathscr{A}$ identical to those described in the proof of Theorem 4, except that there is clock drift with a factor of R; i.e., in the first run $D_r^1(t) = Rt$ for all processors r, while in the second run $D_r^2(t) = R(t - U_N(p, r))$, or equivalently, $D_r^2(t + U_N(p, r)) = Rt$ (note we are using superscripts to distinguish the physical clock functions in the two runs). Take message transmission times to be exactly the same as in the proof of Theorem 4 as measured in real time (i.e., there is no factor of R). The same proof as that in Theorem 4 now shows that these two runs are indistinguishable from the point of view of every processor. Let $C_r^i(t)$ denote processor r's logical clock in run i, $i = 1, 2$. Because of the indistinguishability of these runs, processors must perform the same actions at the same time on their physical clocks. Thus we have that $C_p^1(t) = C_p^2(t)$ for all t, and $C_q^1(t) = C_q^2(t + U_N)$.

By the observations in the first paragraph, we know there is some time t_0 such that $C_p^1(t_0 + U_N) - C_p^1(t_0) \geqslant a(D_p^1(t_0 + U_N) - D_p^1(t_0)) = aRU_N$ (since $D_p^1(t) = Rt$). Thus we get

$$\begin{aligned} aRU_N &\leqslant C_p^1(t_0 + U_N) - C_p^1(t_0) \\ &= (C_p^1(t_0 + U_N) - C_q^1(t_0)) + (C_q^1(t_0) - (C_p^1(t_0)) \\ &= (C_p^2(t_0 + U_N) - C_q^2(t_0 + U_N)) + (C_q^1(t_0) - C_p^1(t_0)) \\ &\leqslant 2\,\text{DMAX}. \end{aligned}$$

The desired result immediately follows. ∎

5. Conclusions

We have presented two flavors of possibility and impossibility results regarding clock synchronization. The first considers the question of when synchronization is achievable in the presence of faults when authentication techniques are not used. We have shown that while certain weak notions of clock synchronization are achievable, regardless of the number of faults (without any communication at all!), linear envelope synchronization does indeed require that less than one-third of the processors be faulty, just as Lamport conjectured [La].

The second type of result considers how *tightly* we can synchronize, as a function of the uncertainty in message transmission time. We have shown that even without

faults and without clock drift, the essential temporal imprecision is always at most equal to the uncertainty, but is always at least half the uncertainty.

The first type of result suggests that in some ways clock synchronization has very much the same flavor as Byzantine agreement: with authentication it can be achieved in the presence of arbitrarily many faults, provided these faults do not disconnect the network (cf. [HSSD]), and without authentication, to achieve LES and to tolerate f faulty processors, there must be at least $3f + 1$ processors in the network, and the connectivity of the network must be at least $2f + 1$.

It is well known that randomness can help in achieving Byzantine agreement, or in reducing the number of rounds required to attain it (see, for example, [Be]). It would be interesting to know if randomness can also help in achieving clock synchronization. For example, can we achieve LES (in some resonable probabilistic sense) using a probabilistic algorithm if one-third or more of the processors are faulty? To get a sense of the problem here, consider the following simple algorithm in the case of three processors. Initially, each processor picks another processor (other than itself). Then, at specified intervals, it asks that other processor for the time on its clock, and sets its own clock to that time. We leave it to the reder to check that this algorithm guarantees LES if there are no faults, and guarantees LES with probability 1/4 if there is one fault (since with probability 1/4 the two correct processors will pick each other at the initial step). We conjecture that there is no probabilistic algorithm that guarantees LES if there are no faults, and guarantees LES with probability greater than 1/2 if there is one fault.

ACKNOWLEDGMENTS

We gratefully acknowledge the two referees of this paper, Jennifer Lundelius and Mike Merritt, for catching a number of errors in a previous version and for their numerous suggestions on improving the readability of the paper. Mike, in particular, convinced us of the need of the assumption of uncertainty of transmission limes in Theorem 2, and of the incorrectness of a theorem claimed in a previous version of this paper, namely that LES is achievable if there is a bound on the rate at which a processor can generate messages. We also than Benny Chor, Ron Fagin, Nancy Lynch, Yoram Moses, and Barbara Simons for a number of helpful comments and criticisms on earlier drafts of this paper.

REFERENCES

[Be] M. BEN-OR, Another advantage of the free choice: Completely asynchronous agreement protocols, *in* "Proceedings of the 2nd ACM Conference on Distributed Computing," 1983, pp. 27–30.

[Do] D. DOLEV. The Byzantine generals strike again, *J. Algorithms* **3** (1982), 14–30.

[DS] D. DOLEV AND H. R. STRONG, Authenticated algorithm for Byzantine agreement, *SIAM J. Comput.* **12** (1983), 656–666.

[FLM] M. FISCHER, N. A. LYNCH, AND M. MERRITT, Easy impossibility proofs for distributed consensus problems, *in* "Proceedings of the 4th ACM Conference on Distributed Computing, 1985," pp. 59–70.

[HMM] J. Y. HALPERN, N. MEGIDDO, AND A. MUNSHI, Optimal precision in the presence of uncertainty, *in* "Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 1985," pp. 346–355.

[HSSD] J. Y. HALPERN, B. B. SIMONS, H. R. STRONG, AND D. DOLEV, Fault-tolerant clock synchronization, *in* "Proceedings of the 3rd ACM Conference on Principles of Distributed Computing, 1984," pp. 89–102.

[La] L. LAMPORT, Unsolved problems, solved problems, and non-problems in concurrency, invited address, *in* "2nd ACM Conference on Principles of Distributed Computing, 1983"; an edited transcript appears *in* "Proceedings of the 3rd ACM Conference on Principles of Distributed Computing, 1984," pp. 1–11.

[LM] L. LAMPORT AND P. M. MELLIAR-SMITH, Byzantine clock synchronization, *in* "Proceedings of the 3rd ACM Conference on Principles of Distributed Computing, 1984," pp. 68–74; a revised and expanded version appears as: Synchronization clocks in the presence of faults, *J. Assoc. Comput. Mach.* **32** (1985), 52–78.

[LSP] L. LAMPORT, R. SHOSTAK, AND M. PEASE. The Byzantine generals problem, *ACM Trans. Prog. Lang. and Systems* **4** (1982), 382–401.

[LL1] J. LUNDELIUS AND N. LYNCH, A new fault-tolerant algorithm for clock synchronization, *in* "Proceedings of the 3rd ACM Conference on Principles of Distributed Computing, 1984," pp. 75–88.

[LL2] J. LUNDELIUS AND N. LYNCH, An upper and lower bound for clock synchronization, *Inform. and Control* **62** (1984), 190–204.

[Ma] K. MARZULLO, "Loosely-Coupled Distributed Services: A Distributed Time System," Ph.D. dissertation, Stanford University, 1983.

[Me] M. MERRITT, private communication, 1985.

[PSL] M. PEASE, R. SHOSTAK, AND L. LAMPORT, Reaching agreement in the presence of faults, *J. Assoc. Comput Mach.* **27** (1980), 228–234.