ON THE POSSIBILITY AND IMPOSSIBILITY OF ACHIEVING CLOCK SYNCHRONIZATION

Danny Dolev

Hebrew University, Givat Ram
91904 Jerusalem, Israel.

Joe Halpern
H. Raymond Strong

IBM Research Laboratory,
San Jose, CA 95193.

ABSTRACT: It is known that clock synchronization can be achieved in the presence of faulty clocks numbering more than one-third of the total number of participating clocks 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 function of real time, then clock synchronization is impossible, without authentication, when one-third or more of the processors are faulty. However, if there is a bound on the rate at which a processor can generate messages, then we show that clock synchronization is achievable, without authentication, as long as the faults do not disconnect the network. Finally, we 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.

1. INTRODUCTION

The problem of achieving clock synchronization in the presence of faults has attracted much attention recently [LM,HSS,DLPSW,Ma]. [HSS] presents an algorithm 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. [LM] presents an algorithm that does not require authentication, but will only work 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, L. Lamport challenged his listeners to provide a proof of impossibility [L]. 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 and some features of the underlying model. For the specific question posed by Lamport with respect to the model of [LM], we provide the expected proof of impossibility.

For simplicity, we assume that each processor has a *duration timer* D (this is usually thought of as the processor's clock) and a designated register TAR called the *time adjustment register*. Correct duration timers exhibit only a bounded rate of drift. The duration timer is never altered by the processor, but the time adjustment register may be altered as a result of a processor's internal operations, receipt of messages, or an indication from its duration timer that a specific amount of time has elapsed. (Altering the TAR is equivalent to starting up a new clock in the notation of [LM] and [HSS]). A processor's logical clock time C is the sum of the TAR and D. Roughly speaking, an 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 P, 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 must 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 duration timer. We eliminate this solution by requiring that the logical clock stay within a *linear envelope* of the duration timer, i.e. by requiring that there be constants a,

b, c, and d such that $aD(t)+c \leq C(t) \leq bD(t)+d$ for all times t. (This condition is satisfied by the clock synchronization algorithms of [HSS] and [LM].) In this case we can show that clock synchronization is not achievable with one-third or more faulty processors (Theorem 2). However, this proof requires that faulty processors have rather unrealistic powers. In fact, linear envelope clock synchronization *is* achievable, even without authentication, provided that there is a bound on the rate at which a processor can generate messages. In Theorem 3 we present an algorithm to do this that is a variant of the algorithm in [HSS].

Finally we consider (Theorems 4 and 5) the degree to which simultaneity can be achieved in a network. We show that for any 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 [HSS] show that this bound can essentially be achieved. These theorems bound the degree to which coordination can be achieved in a given network. If we want two events to happen simultaneously in a network, the best we can do is to guarantee that they will happen at times separated by no more that 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. 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 assume the existence of a Newtonian time frame, not directly observable. As we mentioned in the introduction, each processor has a duration timer D and a time adjustment register TAR. Both of these can be viewed as real-valued functions of Newtonian time. A *correct* duration timer is one that has a bounded rate of drift from real (Newtonian) time. More precisely, there exists a constant $R>1$ such that a correct duration timer satisfies:

$$(1/R)(u-v) < D(u)-D(v) < R(u-v)$$

for all Newtonian times $u > v$. A *correct processor* is one that behaves according to its algorithmic specifications and possesses a correct duration timer. No assumption is made about the behavior of a *faulty* (not correct) processor.

We view TAR(t) as a step function whose value is constant on left-open right-closed intervals, in order to have our results correspond to those of [LM] and [HSS]. All our results would also hold if TAR were continuous. By definition, $C(t)=D(t)+TAR(t)$. Let $TAR(t^+)= \lim TAR(t')$ as t' approaches t from the right, i.e., the value of TAR just after it has been reset at time t. We define $C(t^+)$ similarly.

Processors send messages to each other along the communication links, where a message is just a word over some fixed alphabet Σ. We assume that there are known upper and lower bounds on the transmission and processing of a message from one processor to another. We return to this point in more detail in Section 4.

3. SYNCHRONIZATION WITHOUT AUTHENTICATION

Consider the following definition of clock synchronization:

Weak Clock Synchronization Condition: There exist constants P, B, and S such that, for each correct clock, TAR is constant except that it changes at clock times that are multiples of P by an amount with absolute value less than S, and the difference between the clocks of any correct processors is always bounded by B.

It is left to the reader to verify that any algorithm that achieves the condition expressed by Lamport and Melliar-Smith [LM] also achieves the WCSC.

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 P arbitrarily. Then whenever $C(t)=P$, we set $TAR(t^+)=-D(t)$ (thus making $C(t^+)=0$). Clearly we have $S=B=P$ in this case, since C(t) is always between 0 and P for any correct processor. However, even if we require that C(t) have unbounded range, we still get:

Theorem 1: There is an algorithm which achieves WCSC independent of the number of faults, using no message exchanges, for which C(t) is unbounded.

Proof: The idea is to keep C(t) within the interval $\log(D(t))$ and $\log(D(t))+k$ for some constant k at all times. Since it is easy to deduce from the bounded rate of drift condition that $|\log(D_1(t))-\log(D_2(t)| < 2\log(R)$ for any correct processors p_1 and p_2, it will follow that $|C_1(t)-C_2(t)| < 2\log(R)+k$ at all

times. We proceed as follows. Assume for ease of exposition that $C(0)=D(0)=TAR(0)=0$. Choose $P>0$ arbitrarily. For each time t such that $C(t)=iP$ for some positive integer i, and $\log(D(t))<(i-1)P$ we set $TAR(t^+)=\log(D(t))-D(t)$, thus making $C(t^+)=\log(D(t))$. (Note that the purpose of checking that $\log(D(t))<(i-1)P$, or equivalently, that $C(t)-\log(D(t))>P$, is simply to prevent TAR from being reset infinitely often in a bounded amount of time.) It is easy to check that $0\leq C(t)-\log(D(t))<2P$ for all t, and that TAR is changed by less than 2P at any time. By the arguments made above, it also follows that the logical clocks of correct processors differ by at most $2\log(R)+2P$ at all times. □

It is easy to see that this algorithm would also work if we kept C(t) within any linear function of log(D(t)). In order to achieve an impossibility result, we must strengthen our requirements for clock synchronization somewhat. Essentially we will do this by requiring that C(t) stay within a linear function of D(t).

First note that in the algorithm given in Theorem 1, there may be several times t when $C(t)=iP$ for some i and TAR is reset. If we only allow changes in TAR the first time that C(t) reads iP for any i, than the time between changes can grow unboundedly large, and thus there will be no bound on the difference between the logical clocks of correct processors. A close reading of the clock synchronization condition given in [LM] shows that they indeed require this property. This leads us to the following definition:

Clock Synchronization Condition: There exist constants P, B, and S such that, for each correct clock, TAR is constant except that it can change by an amount with absolute value less than S at a Newtonian time t such that $C(t)=iP\geq 0$ for some integer i and this is the first time that C reads iP (i.e. $C(t')\neq iP$ for all $t'<t$), and the difference between the clocks of any two correct processors is bounded by B.

It is easy to check that the CSC as defined above is equivalent to the condition defined in [LM], in the sense that any algorithm for one can be easily modified to achieve the other.

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

(U,L) Envelope Synchronization: Correct clocks are bounded above by U(D(t)) and bounded below by L(D(t)) and there exists a constant B such that at any Newtonian time the difference between correct clocks is bounded by B. A special case of (U,L) Envelope Synchronization is *Linear Envelope Synchronization*, where U and L are taken to be the linear functions $at+b$ and $ct+d$ respectively, with $c>0$.

Linear Envelope Synchronization guarantees that the time value on a correct clock is within a linear envelope of the time on the duration timer. But since we have assumed that the duration timer is within a linear envelope of real time (bounded by R and 1/R), Linear Envelope Synchronization also implies that the time value on a correct clock is within a linear envelope of real time.

Proposition 1: An algorithm that achieves the Clock Synchronization Condition achieves Linear Envelope Synchronization.

Proof: We leave it to the reader to check that $(P/(P+S))D(t)-S < C(t) < ((P+S)/P)D(t)+S$ for any correct clock C and duration timer D. □

Theorem 1 above shows that (t,log(t)) Envelope Synchronization is achievable. Theorem 2 below will show that Linear Envelope Synchronization is not achievable if one third or more of the processors are faulty. And thus by Proposition 1, the Clock Synchronization Condition is also not achievable if one third or more of the processors are faulty.

Theorem 2: Linear Envelope Synchronization is in general impossible if one third or more of the processors are faulty.

Proof: We first prove a restricted form of Theorem 2 for three processors, one of which is faulty. Suppose that we have an algorithm that achieves Linear Envelope Synchronization of three processors, say p_1, p_2, and p_3, of which one may be faulty. We can suppose that the synchronization guarantees $bD(t)+d\leq C(t)\leq aD(t)+c$, if C and D are the logical clock and duration timer respectively of a correct processor, and that logical clocks of correct processors differ by at most B. Recall that $(1/R)t<D(t)<Rt$ for a correct duration timer D. Our proof is based on the following lemma.

Lemma: If processor p_j is correct (j=1,2, or 3), then for each n, and for every sequence of messages p_k ($k\neq j$) sends, there is a sequence of messages that p_i ($i\neq j,k$) can send p_j that will cause $C_j(t)>bR^{2n}D_j(t)+d-nB$.

Theorem 2 in the case of three processors follows immediately from the lemma. We simply choose n such that $bR^{2n}>a$ (which is always possible since $R>1$). Now suppose that p_1 is faulty, while p_2 and p_3 are correct. From the lem-

ma it follows that p_1 can send messages to p_2 that will cause $C_2(t) > bR^{2n}D_2(t)+d-nB$ for all $t>0$. Since $bR^{2n}>a$, there exists some t' such that $bR^{2n}t+d-nB>at+c$ for all $t>t'$. Since $D_2(t)>(1/R)t$, it follows that $bR^{2n}D_2(t)+d-nB>aD_2(t)+c$ for all $t>Rt'$. Hence $C_2(t)>aD_2(t)+c$ for all $t>Rt'$, which contradicts the fact that the algorithm achieves Linear Envelope Synchronization bounded above by $aD(t)+c$. It is easy to generalize this result to Theorem 2 by considering three sets of processors of equal size such that the duration timers are identical within each set.

To prove the lemma, we proceed by induction on n. By symmetry, we can assume without loss of generality that $i=1$, $j=2$, and $k=3$. For $n=0$, p_1 just follows the algorithm correctly. By hypothesis, no matter what messages p_3 sends, the algorithm achieves $C_2(t)>bD_2(t)+d$.

For $n=1$, first consider the case where p_1 and p_2 are correct, p_3 is faulty, $D_1(t)=Rt$, $D_2(t)=(1/R)t$, and p_3 is not sending any messages to p_1. If p_1 and p_2 both follow the algorithm correctly, then for all $t\geq 0$ we must have $C_2(t) \geq C_1(t)-B > bD_1(t)+d-B = bR^2D_2(t)+d-B$ (using the observation that $D_1(t) = R^2D_2(t)$, by choice of D_1 and D_2). This relationship must hold no matter what messages p_1 sends to p_2, since, by hypothesis, the algorithm tolerates one fault.

Now suppose p_1 sends messages to p_2 as if the situation were that described above. That is, p_1 pretends that its duration timer is running at R^2 the rate of that of p_2 (i.e. $D_1(t)=R^2D_2(t)$) and that p_3 is not sending it any messages, and then does what the algorithm would have said to do if this were the case. No matter what messages p_3 actually sends, p_2 will not be able to distinguish this situation from the one above, so again we will have $C_2(t)>bR^2D_2(t)+d-B$ for all t.

Next, suppose that the lemma holds for $n=m$; we want to show that it also holds for $n=m+1$. Again we first consider the situation where p_1 and p_2 are correct, p_3 is faulty, $D_1(t)=Rt$, $D_2(t)=(1/R)t$, and p_3 is sending p_1 messages that cause it to have $C_1(t)>bR^{2m}D_1(t)+d-mB$. (By the induction hypothesis this is always possible no matter what messages p_2 sends p_1 and p_3, although of course the messages that p_3 sends p_1 may depend on the messages that p_1 sends to p_3, which in turn may depend on the messages that p_2 sends to p_1.) Now if p_1 and p_2 just follow the algorithm correctly, an analogous argument to the one made above shows that we must have $C_2(t)>bR^{2(m+1)}D_2(t)+d-(m+1)B$.

Now suppose again that p_1 sends messages to p_2 as if the situation were that described above. That is, p_1 pretends that its duration timer is running at R^2 the rate of that of p_2 and p_3 is sending it messages that cause $C_1(t) > bR^{2m}D_1(t)+d-mB$. Again, p_2 will not be able to distinguish this situation from the one above, so again, no matter what messages p_3 sends, we will have $C_2(t) > bR^{2(m+1)}D_2(t)+d-(m+1)B$.

This completes the proof of the lemma, and with it the proof of Theorem 2. □

We remark that by combining the proof of Theorem 2 with some of the techniques of [D], we can also show that clock synchronization is impossible if the connectivity of the network is $< 2t+1$ and there are t or more faulty processors. We omit details here.

The proof of Theorem 2 requires that a faulty processor have some rather unreasonable powers. In order to act as if it is receiving messages from another processor that are causing it to set its clock running faster than $bR^{2n}+d-nB$, it might have to forge signatures, which is very unlikely if we have a good signature scheme. It might also have to generate messages at a very great speed. Indeed, as the following theorem shows, if there is a bound on the rate at which messages can be generated, then there exists an algorithm that can achieve the Clock Synchronization Condition, and hence Linear Envelope Synchronization.

Theorem 3: If there is a bound on the rate at which messages can be generated or there is a protocol for signing unforgeable signatures that can be authenticated, then the Clock Synchronization Condition can be achieved as long as the faults do not disconnect the network.

Proof: In [HSS], there is an algorithm that can easily be modified so as to achieve the Clock Synchronization Condition, but requires a protocol for signing unforgeable signatures that can be authenticated. We now show how to modify that algorithm so that it works even without authentication, as long as there is a bound on the rate at which messages can be transmitted. We proceed as follows.

We assume that each processor p_i has an internal variable ET_i (expected time of next synchronization) and has access to a continuously updated variable C_i which gives its logical clock time. (We will omit the subscript i when it is clear from context. As explained in Section 2, C is the sum of the times on a processor's duration timer and its time

adjustment register. C can be reset (by changing the time adjustment register TAR) and read. There is a global constant PER (period of resynchronization) common to all processors. We further assume ET is initially PER for all processors, and that initially C < PER. Finally, we assume that if p_i and p_j are neighbors in the network (i.e are connected by direct link) then as long as they and the link between them are nonfaulty, then messages from p_i to p_j are received in the order that they are sent. (If the communication network does not have this property, we can simply number all messages consecutively and ignore a message numbered n until all messages with a lower number have been received.)

The algorithm consists of two tasks that are run independently and concurrently. The first task consists of one instruction:

```
if C=ET
   then ET:=ET+PER; send to all neighbours
      a message saying "The time is ET"
fi
```

The second task describes what to do upon the receipt of a message saying "The time is T":

```
if a message saying "The time is T" is received,
   and T=ET and C<ET
   then C:=ET; ET:=ET+PER; send to all neighbors
      a message saying "The time is ET"
   else do nothing
fi.
```

To prove the correctness of the algorithm above we proceed as follows. Fix a network G and suppose there exists a set F of faulty nodes and links in G which do not disconnect G. Let G/F be the surviving graph (G with all the faults in F removed) and let DIAM(G/F) be its diameter; i.e. the worst case distance between two nodes in G/F. Let TDEL be the worst-case time for a message to be broadcast by one nonfaulty processor to all its nonfaulty neighbors, and suppose that the bound on the rate of message transmission is m messages per unit time. We now show that if p_i and p_j are nonfaulty, then $|C_i(t)-C_j(t)| \leq m|G|DIAM(G/F)(TDEL)(PER)$, where $|G|$ is the number of nodes in G.

Using techniques of [HSS] one can show that the algorithm has the following properties:

(a) for any nonfaulty processor p_i, we always have

$$ET\text{-}PER \leq C \leq ET$$

(b) if p_i is nonfaulty, then, for all k, p_i will broadcast the message "the time is (k+1)PER" to its neighbors after it has broadcast the message "the time is kPER", but at most R(PER) after this time (recall that R is an upper bound on the rate of drift of the duration timer from real time).

(c) if p_i and p_j are neighbors in G and are nonfaulty, and p_j receives a message from p_i saying "the time is kPER", then either ET_j=kPER or $C_j \geq$ kPER.

(d) all nonfaulty processors will set their clocks to kPER within DIAM(G/F)TDEL of the time that the first nonfaulty processor does so.

To complete the proof of the result, note that from the assumption that m is an upper bound on the number of messages that can be generated by one processor in one time unit, it follows that in DIAM(G/F)TDEL time units, at most m|G|DIAM(G/F)TDEL messages can be generated in the network. Any message received by a processor can force it to push its clock forward by at most PER. Using part (d) it is now easy to see that at all times the clocks of nonfaulty processors differ by at most m|G|DIAM(G/F)(TDEL)(PER). □

4. LOWER BOUNDS ON SYNCHRONIZATION

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

To investigate this issue more carefully, we turn our attention from the tightness of synchronization along the 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 that achieves linear envelope synchronization 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.

For this analysis, we will consider a more detailed model of transmission and processing delay. Fix a communication network G. Processors can only communicate over the links in the network. We assume that there are known upper and lower bounds on the time to transmit and process a message from p to q if they are joined by a direct link. Thus, we can define the following functions on processors p and q such that there is a direct link between p and q:

$H_G(p,q)$ = upper bound on transmission and processing time for messages between p and q.

$L_G(p,q)$ = lower bound on transmission and processing time for messages between p and q.

$V_G(p,q)$ = variation in transmission and processing time for messages between p and q
$= H_G(p,q)-L_G(p,q)$.

We extend H_G, L_G, and V_G so that they apply to all pairs of processors by setting $H_G(p,q) = L_G(p,q) = V_G(p,q) = \infty$ for processors p, q such that there is no direct link from p to q. We now extend V_G so that it also applies to sequences of processors. For any sequence of processors $\pi = p_0,p_1,\ldots,p_n$, let $V_G(\pi)$ be the sum of the values $V_G(p_i,p_{i+1})$, for i from 0 to n-1. Finally, let $U_G(p,q)$ (the *uncertainty* in transmission time from p to q) $= \min\{V_G(\pi) \mid \pi$ is a sequence of processors starting with p and ending with q$\}$, and let $U_G = \max\{U_G(p,q) \mid$ p,q are processors in G$\}$.

For ease of exposition in what follows, we will assume that each processor has a special register which initially contains the value 0. At some point the value must be changed to 1. The problem is to obtain an algorithm which guarantees that all processors change the value to 1 at as close to the same real time as possible. The algorithm must be deterministic, in that what each processor does can depend only on its duration timer and message history. To make this precise, note that there is some inherent nondeterminism in the system because of the uncertainty of the transmission time of messages and the rate of drift of clocks. Let a *run* r of algorithm A in network G be a particular choice of transmission times for each message transmitted and a choice for the rate of drift of each processor's duration timer, subject to the constraints discussed above. If a given processor p performs a certain action a at a given time T on its duration timer (i.e. at real time t such that $D_p(t)=T$) in a certain run r, then action a will be performed at time T on its duration timer in any other run r′ with the same message history; i.e. any run where messages from other processors to p arrive at the same time on p's duration timer. Clearly the time at which a processor changes the value in its special register from 0 to 1 will depend on the run. The essential temporal imprecision inherent in a particular algorithm 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 runs. The essential temporal imprecision in a network is the minimum essential temporal imprecision over all possible algorithms. More formally, given an algorithm A, processors p and q in G, and run r define

$\Delta_{G,A}(p,q,r)$ = the absolute value of the difference of the real times at which processors p and q change the value of their special register in run r of clock synchronization algorithm A.

$\Delta_{G,A}(p,q) = \max_r\{\Delta_G(p,q,r)\}$.

$\Delta_G(p,q) = \min_A\{\Delta_{G,A}(p,q)\}$.

$\Delta_{G,A} = \max_{p,q}\{\Delta_{G,A}(p,q)\}$.

$\Delta_G = \min_A\{\Delta_{G,A}\}$.

Theorem 4: For all communication networks G and all processors p, q in G, we have $\Delta_G(p,q) \geq U_G(p,q)/2$; i.e. the imprecision is at least half the uncertainty.

Proof sketch: Fix network G and processors p and q in G. We consider two runs which, as we shall show, are indistinguishable from the point of view of any processor. In the first run, all processors are started at the same time, with their duration timers set to 0 and proceeding at exactly the rate of real time (i.e. there is no drift). If there is a link from processor r to processor r′ in G, then messages from r to r′ take time $L_G(r,r')+\max(U_G(p,r)-U_G(p,r'),0)$. We leave it to the reader to check that $U_G(p,r)-U_G(p,r') \leq V_G(r,r')$, so a message from r to r′ can indeed take this length of time.

In the second run, we will start processor p first, and start each processor r at real time $U_G(p,r)$ later than p. Again, each processor's duration timer reads 0 when it is started and proceeds at exactly the rate of real time. If r and r′ are joined by a link in G, then messages from processor r to r′ take time $L_G(r,r')+\max(U_G(p,r')-U_G(p,r),0)$ to arrive. Again it is easy to check that this meets the constraints above, and that no message will reach a processor before it has been started. We now show that these two runs are indistinguishable from the point of view of any processor; i.e. they produce the same message history.

Suppose r and r′ are joined by a link in G, and r sends r′ a message when r's duration timer reads T. We first consider the case where $U_G(p,r) \geq U_G(p,r')$. In the first run, this message will arrive at r′ in time $L_G(r,r')+U_G(p,r)-U_G(p,r')$, when the duration timer of r′ reads $T+L_G(r,r')+U_G(p,r)-U_G(p,r')$. In the second run, this message will arrive at r′ in time $L_G(r,r')$, but again the duration timer of r′ will read $T+L_G(r,r')+U_G(p,r)-U_G(p,r')$, since r′ is started $U_G(p,r)-U_G(p,r')$ ahead of r. The argument in the case where $U_G(p,r)<U_G(p,r')$ is similar, and is omitted here.

Because messages are being sent and received at the same time on each processor's duration timer in both runs, processors will perform the same action at a given time on their duration timers in both runs. Suppose processor p changes the value of its special register at time T_1 on its duration timer, while processor q changes the value at time T_2 on its duration timer. Let t_1 and t_2 be the real times that the duration timers of p and q read T_1 and T_2 respectively in the first run. Note that in the second run, processor p's duration timer still reads T_1 at t_1, but processor q's duration timer reads T_2 at $t_2+U_G(p,q)$ (since processor q was started $U_G(p,q)$ later in the second run). It is now easy to see that $\max(|t_2-t_1|, |t_2+U_G(p,q)-t_1|) \geq U_G(p,q)/2$, which gives us our result. □

Remarks: Note that the lower bound holds even if there are no faults in the network. We can also prove a version of this result in which all processors all start the second run at the same time with their duration timers synchronized, but then the duration timers of some processors drift so that p's duration timer is $U_G(p,r)$ ahead of that of r. From this point we can essentially repeat the proof above.

Corollary 1: For all communication networks G, we have $\Delta_G \geq U_G/2$.

Proof: Note that $\Delta_G \geq \max_{p,q}\Delta_G(p,q)$, 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. □

Corollary 2: For any clock synchronization algorithm in which there are at least two values which all logical clocks take at some time, there is a value T such that the real time at which two processors first read T in some run differs by $\geq U_G/2$.

Theorem 5: For all $\epsilon > 0$, there exist a network G such that $\Delta_G \geq U_G-\epsilon$.

Proof: This result follows from a result of [LL], where it is shown that if G is a completely connected graph with n nodes, such that for all processors p, q we have $V_G(p,q)=\delta$, so that $U_G=\delta$, then $\Delta_G=((n-1)/n)\delta=U_G-(1/n)\delta$. This clearly gives us the desired result. □

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

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

Proof: The clock synchronization algorithm of [HSS] 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 read T' differ by at most U_G in any run. Each processor can thus use its logical clock to decide when to change the value in its special register. □

We remark that the algorithm of [HSS] works even in the presence of faults. However, we seem to require authentication if we are to keep the imprecision no greater than the uncertainty in the presence of faults.

Corollary 3: For all communication networks G, $\Delta_G \leq U_G$.

Finally, we observe that we can easily translate the results of Theorem 4 to bounds on the tightness of synchronization for algorithms which achieve Linear Envelope Synchronization.

Theorem 7: If A achieves linear envelope synchronization in communication network G with bound B and lower envelope $cD(t)+d$, then $B \geq cRU_G/2$.

Thus we can see that there is a tradeoff between tightness of clock time synchronization and tightness of the linear envelope.

REFERENCES

[D] D. Dolev, The Byzantine generals strike again, *Journal of Algorithms*, 3, 1982, pp. 14-30.

[DLPSW] D. Dolev, N. A. Lynch, S. Pinter, E. Stark, and W. Weihl, Reaching approximate agreement in the presence of faults, Proceedings of the 3rd Annual IEEE Symposium on Distributed Software and Databases, 1983 (also available as MIT/LCS/TM-251).

[DS] D. Dolev and H. R. Strong, Authenticated algorithms for Byzantine agreement, *SIAM J. of Computing*, to appear, 1983.

[HSS] J. Y. Halpern, B. B. Simons, and H. R. Strong, An efficient fault-tolerant algorithm for clock synchronization, IBM RJ4094, 1983.

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

[LSP] L. Lamport, R. Shostak, and M. Pease, The Byzantine Generals problem, *ACM Trans. on Prog. Lang. and Systems* 4:3, 1982, 382-401.

[LL] J. Lundelius and N. Lynch, Synchronizing clocks in a distributed system, unpublished manuscript, 1984.

[Ma] K. Marzullo, Loosely-coupled distributed services: a distributed time system, Ph.D. dissertation, Stanford University, 1983.

[PSL] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *JACM* 27:2, 1980, 228-234.