JOURNAL OF COMPLEXITY 1, 170–196 (1985)

Optimal Precision in the Presence of Uncertainty*,†

JOSEPH Y. HALPERN, NIMROD MEGIDDO, AND ASHFAQ A. MUNSHI

IBM Research Laboratory, San Jose, California 95193

We consider the problem of achieving coordinated actions in a real-time distributed system. In particular, we consider how closely (in terms of real time) processors can be guaranteed to perform a particular action, in a system where message transmission is guaranteed, but there is some uncertainty in message transmission time. We present an algorithm to achieve optimal precision in arbitrary networks. In networks where clocks run at the rate of real time, the optimal precision achievable in a network is exactly how tightly clocks can be guaranteed to be synchronized.

1. INTRODUCTION

Achieving coordinated actions in a real-time distributed system often requires that processors perform actions at roughly the same time. Suppose the processors in a system want to perform a particular action (such as flipping a bit) simultaneously. How closely (in terms of real time) can they be guaranteed to perform this action?

Of course, the answer to this question depends on the model of computation. Here we assume that processors communicate by sending messages over links in a network. Message delivery is guaranteed, but processors can neither control nor observe the amount of time it takes messages to be delivered along a given link. All they know are lower and upper bounds on these message delivery times. Each processor has its own local clock, but there is no universal clock available to the processors. For the moment, we will also assume that each processor's local clock runs at the rate of real time, and that there are no faulty processors.

The *(essential temporal) imprecision*—the degree of simultaneity that can be guaranteed—is a fundamental property of a communication network.

* Presented at the Symposium on Complexity of Approximately Solved Problems, April 17, 1985.

† A preliminary version of this paper appears in the Proceedings of the 17th Annual ACM Symposium on Theory of Computing.

0885-064X/85 $3.00

Clearly it is a function of the network topology and the uncertainty in message delivery time (i.e., the difference between the upper and lower bounds) between the processors and the topology of the network. General lower and upper bounds on imprecision are given by Dolev *et al.* (1984). Tight bounds were heretofore known in only one special case: that of a completely connected network with clocks running at the rate of real time and an uncertainty of ϵ over all links. Lundelius and Lynch (1984b) show that in this case the imprecision is precisely $\Delta = ((n - 1)/n)\epsilon$. They also present a simple algorithm to synchronize clocks to within Δ, showing this degree of simultaneity is achievable.

In this paper we present an algorithm to achieve optimal precision in arbitrary networks. We can view the processors as playing a game against an adversary ("nature" or "the system"): the processors would like to perform a particular action as closely together in real time as possible, while the adversary tries to prevent this by appropriately choosing the initial difference between the processors' clocks and the message transmission time for each message, subject to the given upper and lower bounds. We give a technique that guarantees that the worst-case difference between when the first and last processors perform the action is optimal given the choices made by the adversary. Of course, if the adversary in turn makes optimal choices, this difference is precisely the imprecision of the network. The technique involves the solution of shortest path and related network optimization problems, and is easily implementable in polynomial time.

Of course, questions about imprecision are closely related to questions about how tightly one can synchronize clocks (cf. the clock synchronization algorithms of Lamport and Meilliar-Smith, 1984; Halpern *et al.*, 1984; Lundelius and Lynch, 1984a). Certainly if we can completely synchronize clocks then we can achieve simultaneity (every processor flips the bit when its clock reads 11 AM); conversely, if we can achieve simultaneity (and all clocks run at the same rate), then we can completely synchronize clocks (all clocks are set to 11 AM when they flip the bit). Indeed, if clocks are guaranteed to run at the rate of real time and clocks can only be adjusted a bounded number of times, then the imprecision of the network is easily seen to be equal to the tightness of synchronization achievable (this point is discussed in more detail in Section 2). Of course, in practice, a processor's physical clock does drift away from real time; clock synchronization algorithms are used to synchronize a processor's *logical* clock. As pointed out by Dolev *et al.* (1984), if logical clocks are constrained to run within a linear envelope of real time, then lower bounds on imprecision immediately give lower bounds on the degree of synchronization that can be achieved by a clock synchronization algorithm.

The rest of this paper is organized as follows. The model is formally introduced in Section 2. In Section 3 we give some insight into the difficulty

of the problem by means of some simple examples. In particular we show that in a three-processor network where the uncertainty in transmission times between pairs of processors is a, b, and c, with $c \geq \max(a, b)$, the imprecision is exactly $\max(\frac{1}{3}(a + b), \frac{1}{2}c)$. We also show how the "many-scenarios" techniques of Dolev *et al.* (1984), and Lundelius and Lynch (1984b) can be generalized to prove the lower bound. In Section 4 we prove our main result, presenting the strategy that guarantees that the processors achieve the optimal degree of simultaneity, given the adversary's moves. In Section 5 we show that the processors can do no better using probabilistic algorithms than they can by using deterministic algorithms. Indeed, we show that there is a probabilistic strategy that nature can follow such that for *any* probabilistic strategy the processors follow, the expected precision attained by the processors is no better than the optimal precision attainable by a deterministic strategy. In Section 6, we reduce the problem of computing the imprecision of an arbitrary (connected) network to that of computing the imprecision of a completely connected network where the lower bound on message transmission time is 0. We also show, somewhat surprisingly, that the imprecision in a directed network is the same as the imprecision in an undirected network where the uncertainty in transmission time of messages between processors p and q is the minimum of the uncertainty in transmission time of messages between p and q and between q and p. In Section 7 we show by a simple trick that in a precise sense fault tolerance results in no loss of precision and conclude in Section 8 with a number of open problems.

2. The Model

The model presented here is essentially that of Dolev *et al.* (1984), slightly generalized to allow one-way communication. We view a communication network as a directed graph, where nodes in the graph represent processors while edges represent communication links. We assume that the graph is *strongly connected* (for all nodes p and q, there is some path from p to q), but not necessarily completely connected. Associated with the edge from p to q are two numbers $H(p, q)$ and $L(p, q)$ that represent upper and lower bounds on the time to transmit (and process) a message from p to q. More formally then, we take a communication network C to be a triple (G, H, L), where $G = (V, E)$ is a directed graph, and H, L are functions from E to the non-negative reals such that $H(p, q) \geq L(p, q)$.

We assume the existence of an external time frame, not directly observable. Each processor has a *physical clock* (or *duration timer*) D which can be viewed as a real-valued function of real time. Like Lundelius and Lynch (1984b), we will assume that physical clocks run at the rate of real time; thus $D(t) = t + D(0)$ for all times t.

We assume that each processor has a local variable TAR (for *time adjust-*

ment register) which provides a correction to its physical clock to give the logical clock reading. We can also view TAR as a real-valued function of real time. We define $C(t) = D(t) + \mathrm{TAR}(t)$. Thus we can think of $C(t)$ as representing a processor's logical clock time at real time t.

A processor p's *message history* at real time t consists of a finite sequence of tuples of the form $\langle q, m, T, y \rangle$, where q is a processor, m is a message, T is a time on p's physical clock, and y is either **received** or **sent.** Intuitively, at real time t, there is one such tuple for each message that p has received or sent up to, but not including, time t, where q indicates that the message was received from or sent to processor q, m is the value of the message, and T indicates that $D_p(t') = T$ at the real time t' $(< t)$ when the message arrived or was sent. A *deterministic algorithm* is one where the *actions* performed by a processor at a given real time t (i.e., the messages it sends and the updates that it makes to internal variables like TAR) depend only on its physical clock reading $D(t)$ and its message history at time t (and thus only on messages received up to, but not including, time t). For now we restrict our attention to deterministic algorithms. In Section 5 we extend our results to probabilistic algorithms, where a processor's actions may also depend on the result of random coin tosses.

We next define the notion of imprecision. For ease of exposition, 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. We want to compute how tightly this can be guaranteed to be done (in terms of real time). Of course, one way of doing this is to synchronize logical clocks as tightly as possible, and then change the value in the register at a particular clock reading. Note that if physical clocks run at the rate of real time, then the precision is exactly the optimal clock synchronization achievable; many of our proofs make use of this fact. In general, we define the *essential temporal imprecision inherent in running algorithm $\mathcal{A}$ in network C*, $\Delta_{C,\mathcal{A}}$, as the worst-case difference in the real times that two processors change the value of their special register, when all processors in C run algorithm $\mathcal{A}$. The essential temporal imprecision, or just *imprecision*, Δ_C, in a network C is the minimum of $\Delta_{C,\mathcal{A}}$ over all algorithms $\mathcal{A}$.

More formally, define a *scenario* to be a specification of the algorithm run by each processor, the function describing the physical clock of each processor, and a choice of message transmission time for each message (subject to the constraints given). A *run* of an algorithm $\mathcal{A}$ in a communication network C is just a scenario in which all the processors in C run $\mathcal{A}$. Following Dolev *et al.* (1984), given an algorithm $\mathcal{A}$, a communication network C, two processors p and p' in C, and a run r of $\mathcal{A}$ we define

$\Delta_{C,\mathcal{A}}(p, p', r) =$ the difference between the real times when p and p' change the value of their special register in run r of $\mathcal{A}$,

$$\Delta_{C,\mathcal{A}}(r) = \max_{p,p'}\{\Delta_{C,\mathcal{A}}(p, p', r)\},$$
$$\Delta_{C,\mathcal{A}} = \max_{r}\{\Delta_{C,\mathcal{A}}(r)\},$$
$$\Delta_{C} = \min_{\mathcal{A}}\{\Delta_{C,\mathcal{A}}\}.$$

Thus $\Delta_{C,\mathcal{A}}(r)$ is the the closest real-time synchronization that can be guaranteed any pair of processors in run r of algorithm $\mathcal{A}$, $\Delta_{C,\mathcal{A}}$ is the inherent imprecision of algorithm $\mathcal{A}$ in network C (the worst-case real-time difference between when two processors in C perform a given action over all runs of $\mathcal{A}$), and Δ_C is the imprecision in C: the tightest coordination in C that can be guaranteed by *any* algorithm.

As we mentioned in the Introduction, there is a close relationship between the imprecision in a network and how tightly logical clocks can be synchronized. Suppose we have an algorithm that can compute a value of TAR for each processor that guarantees that processors' logical clocks are synchronized to within a specified value, say ϵ, and suppose that processors' physical clocks run at the rate of real time (so that once we get logical clocks to within ϵ, they stay within ϵ). Then we can clearly use the processors' logical clocks to decide when to change the value of the special register, so the imprecision is $\leq\epsilon$. Conversely, if we have an algorithm that achieves an imprecision of ϵ, then we can set all logical clocks to some agreed-upon value, say 0, at the point when the value of the special register is changed (so that if this happens at real time t for processor p, processor p sets $\mathrm{TAR}_p(t) = -D_p(t)$, thus making $C_p(t) = 0$). If we assume that physical clocks run at the rate of real time, then this algorithm guarantees that logical clocks stay synchronized to within ϵ.

We make use of this connection between clock synchronization and imprecision throughout this paper. In particular, we give algorithms that compute a value of TAR for each processor such that logical clocks are synchronized to within some predetermined ϵ. We then use these logical clocks in our algorithm for optimal precision.

However, note that in practical applications we cannot assume that physical clocks run at the rate of real time. Thus, in practical clock synchronization algorithms (Lamport and Melliar-Smith, 1984; Halpern *et al.*, 1984; Lundelius and Lynch, 1984a), logical clocks are periodically resynchronized (i.e., TAR is adjusted periodically, rather than just a bounded number of times). And from the fact that logical clocks differ by at most ϵ at any given real time, it does *not* immediately follow that by using logical clocks we can guarantee that the imprecision is less that ϵ. Although at any real time logical clocks may be no more than ϵ apart, it may still be the case that the real times when logical clocks read a given time, say 11 PM, are arbitrarily far apart.[1]

[1] Actually, the clock synchronization algorithms of Lamport and Melliar-Smith (1984), Halpern *et al.* (1984), and Lundelius and Lynch (1984a) also guarantee upper bounds on how

It is exactly because of this that we consider the question of imprecision in a network to be more fundamental than the question of how tightly clocks can be synchronized. Ultimately, we want to coordinate real-time actions, not synchronize logical clocks (see Dolev *et al.* (1984) for further discussion on this point).

3. Some Examples

In order to give the reader insight into the difficulties involved in computing Δ_C we consider some examples in this section.

The first example is a network C_2 consisting of two processors p and q, with $L(p, q) = L(q, p) = 0$ and $H(p, q) = H(q, p) = a$. Here we have

PROPOSITION 3.1. $\Delta_{C_2} = \frac{1}{2}a$.

Proof. For the upper bound we proceed as follows. Processor p sends a message to q; q changes the value of the special register when it gets the message, while p does so $\frac{1}{2}a$ after the message is sent. It is now easy to check that these values are changed within $\frac{1}{2}a$ of each other.

Results of both Dolev *et al.* (1984) and Lundelius and Lynch (1984b) show that this in fact is a tight bound. We briefly review details of the lower bound argument here. Fix an algorithm $\mathcal{A}$. We consider two runs of $\mathcal{A}$. In the first, the processors' physical clocks are identical (i.e., $D_p(t) = D_q(t)$), messages from p to q take time a, while messages from q to p take 0 time. In the second, processor q's physical clock is a ahead of p's ($D_q(t) = D_p(t) + a$), messages from p to q take 0, while messages from q to p take a. It is now straightforward to check that these two runs are indistinguishable from the point of view of both p and q; i.e., they both get the same messages at the same time on their physical clocks in both runs. (Although the details here are straightforward, a careful proof is not completely trivial; see Lemma 4.1 below for a generalization of this result.)

Since the runs are indistinguishable, the processors must perform a given action at the same time on their physical clocks in both runs. Suppose p changes the value of its special register at time T_1 on its physical clock, while q changes the value at time T_2 on its physical clock. Let t_1 and t_2 (resp. u_1 and u_2) be the real times that the physical clocks of p and q read T_1 and T_2, respectively, in the first (resp. second) run. By definition, $\Delta_{C_2,\mathcal{A}} \geq \max(t_2 - t_1, t_1 - t_2, u_2 - u_1, u_1 - u_2)$. Since the processors' physical clocks are identical in the first run, while q's physical clock is a ahead of p's in the second run, we must have $t_2 - t_1 = a + u_2 - u_1$. Thus if $(u_1 - u_2) \leq \frac{1}{2}a$, then $(t_2 - t_1) \geq \frac{1}{2}a$. The result is now immediate. ■

far apart the real times are that logical clocks read a given time T, so that logical clocks can be used for real-time coordination. However, it is still not the case that the tightness of synchronization is equal to the imprecision.

Although we have assumed $L(p, q) = 0$ here, an easy extension of the argument above can be used to show that the imprecision in the general two-processor case is $\frac{1}{2}U$, where $U = \min(H(p, q) - L(p, q), H(q, p) - L(q, p))$. In fact, as we show in Section 6, the imprecision always depends only on the uncertainty, as long as physical clocks run at the rate of real time.

We can extend the arguments of Proposition 3.1 to get tight bounds on the imprecision for tree-like networks and general upper bounds for the imprecision of all networks. For simplicity, we will limit the discussion below to *special* networks where the lower bound on transmission time over any link is 0 (i.e., $L(p, q) = 0$ for all edges (p, q)), and the network is symmetric in that $H(p, q) = H(q, p)$ for all edges (p, q). (We remark that the discussion in Section 6 below shows that these restrictions can be made without loss of generality. More precisely, we show in Section 6 that given any communication network we can always find, in linear time, a special network with the same imprecision.)

Recall the standard notions of *diameter* and *radius* from graph theory: the diameter of a graph is the distance between the two nodes that are furthest apart, while the radius is the minimum over over all nodes p of the maximum (over all nodes q) of the distance from p to q. Given a special network $C = (G, H, L)$, we define its *diameter* (resp. *radius*) *with respect to the uncertainty* to be the diameter (resp. radius) of the graph G where the distance between adjacent nodes p and q is $H(p, q)$.

PROPOSITION 3.2. *Suppose $C_T = (G, H, L)$ is a special tree-like network. Then $\Delta_{C_T} = \frac{1}{2}D$, where D is the diameter of C_T with respect to the uncertainty.*

Proof. The proof is easy. First, a straightforward extension of the lower bound argument in Proposition 3.1 gives us a lower bound of $\frac{1}{2}D$. For the upper bound, we give an algorithm to compute a value of TAR for each processor that guarantees that all logical clocks are synchronized to within $\frac{1}{2}D$. As we remarked in Section 2, this suffices to show that $\Delta_C \leq \frac{1}{2}D$. For this proof, we assume all clocks mentioned are logical clocks, not physical clocks.

If there are two nodes, we can synchronize the clocks of the two processors, say p and q, by essentially the same process as described in Proposition 3.1: p sends q a time-stamped message, say with time T. When q receives the message, it sets its clock to $T + \frac{1}{2}D$. Since p's clock reads some time between T and $T + D$ when q receives the message, we are done.

In the general case, consider any longest path P in the tree. Let (p, q) be a link on P that contains the midpoint of P. By removing the link (p, q) from the tree we are left with two tree-like subnetworks: C_p, containing p, and C_q, containing q. Let L_1, L_2, and L_3 denote the lengths of the three portions of P corresponding to C_p, the link (p, q), and C_q, respectively. Thus $L_1 + L_2 + L_3 = D$, $L_1 \leq \frac{1}{2}D$, and $L_3 \leq \frac{1}{2}D$.

It is easy to see that the path from p to any node of the subnetwork C_p has length at most L_1. Thus, by an easy extension of the arrangement presented for the case of two processors, we can show that the clocks of all processors in C_p can be synchronized to within $\frac{1}{2}L_1$ of that of p. Similarly, the clocks of all processors in C_q can be synchronized to within $\frac{1}{2}L_3$ of that of q. Thus, we start by p and q synchronizing their clocks to within $\frac{1}{2}L_2$ using the simple algorithm described above. We then synchronize the clocks of all processors in C_p to within $\frac{1}{2}L_1$ of p's clock, and the clocks of processors in C_q to within $\frac{1}{2}L_3$ of that of q. It is easy to see that all the clocks of processors in C_p are within at most $L_1 \leq \frac{1}{2}D$ of each other, the clocks of processors in C_q are within at most $L_3 \leq \frac{1}{2}D$ of each other, and it is also easy to check that the clock of a processor in C_p is at most $\frac{1}{2}(L_1 + L_2 + L_3) = \frac{1}{2}D$ of a processor in C_q. This observation completes the proof. ■

PROPOSITION 3.3. *For any special communication network C, $\Delta_C \leq R$, where R is the radius of C with respect to the uncertainty.*

Proof. Let p_0 be the "center" of C, i.e., the processor such that the maximum distance from p_0 to q in the graph with edge weights $H(p, q)$ is R. By arguments similar to those used in Proposition 3.2, we can easily show that the logical clock of every processor can be synchronized to within $\frac{1}{2}R$ of that of p_0. Thus, all logical clocks in the network can be synchronized to within R. ■

We next consider a fully connected communication network C_3 consisting of three processors p_1, p_2 and p_3. Assume $L(p_i, p_j) = 0$, $H(p_1, p_2) = H(p_2, p_1) = a$, $H(p_2, p_3) = H(p_3, p_2) = b$, and $H(p_1, p_3) = H(p_3, p_1) = c$, where $c \geq \max(a, b)$ and $c \leq a + b$ (see Fig. 1).

PROPOSITION 3.4. $\Delta_{C_3} = \max(\frac{1}{3}(a + b), \frac{1}{2}c)$.

Proof. The fact that $\Delta_{C_3} \geq \frac{1}{2}c$ follows immediately from the arguments used in Proposition 3.1. A "three-scenario" argument along similar lines can be used to show that $\Delta_{C_3} \geq \frac{1}{3}(a + b)$. We omit details here. Thus $\Delta_{C_3} \geq \max(\frac{1}{3}(a + b), \frac{1}{2}c)$.

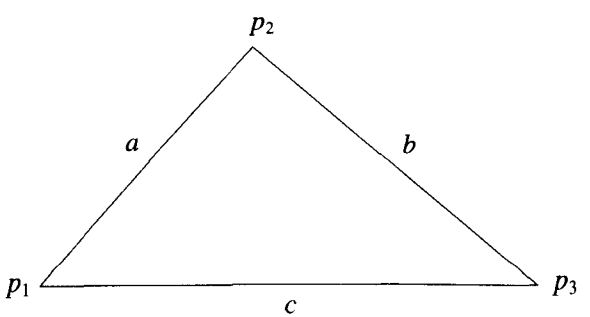

FIG. 1.

For the upper bound, we give an algorithm that synchronizes (logical) clocks to within $\max(\frac{1}{3}(a + b), \frac{1}{2}c)$. (We again assume that all clocks mentioned throughout this proof are logical clocks, rather than physical clocks.)

The first step is to synchronize the clocks of p_1 and p_3 so that they are within $\frac{1}{2}c$ of each other. This can be done easily as described in the proof of Proposition 3.2 above.

Processors p_1 and p_3 next send time-stamped messages to p_2. For ease of exposition, let us assume that they are sent at the same time T on their clocks. Suppose that processor p_2 receives p_1's message at time T_1 and p_3's message at time T_3 on its (p_2's) clock. p_2 knows that p_1's clock reads T at some time in the interval $[T_1 - a, T_1]$ on its (p_2's) clock, and that p_3's clock reads T at some time in the interval $[T_3 - b, T_3]$. Suppose $T_1 - a \leq T_3 - b$ and $T_1 \leq T_3$ (see Fig. 2); a completely analogous argument works in the other three cases. Note that if p_2 sets its clock to read T at the midpoint of the interval $[T_1 - a, T_3]$, it would be at most $\frac{1}{2}(T_3 - T_1 + a)$ away from both p_1 and p_3. It may be able to do even better by telling p_3 to set its clock back by an amount k. In order to reduce the total span of the intervals where p_1's and p_2's clocks read T, k must be chosen so that $0 \leq k \leq \min(T_3 - T_1, T_3 - b - (T_1 - a))$ (there is no advantage in shifting any more than the amount needed to make one interval completely contain the other). After shifting by k, p_3's clock will read T at some point in the interval $[T_3 - b - k, T_3 - k]$. Then by setting its clock so that it reads T at the midpoint of the interval $[T_1 - a, T_3 - k]$, p_2 can guarantee that its clock differs from those of p_1 and p_3 by at most $\frac{1}{2}(T_3 - k - T_1 + a)$.

Shifting p_3's clock back can also increase the difference in the readings of p_1's and p_3's clocks, but this is only possible if initially p_1's clock read T earlier (in real time) than that of p_3. But we know that p_1's clock could have reat T at most e earlier than that of p_3, where $e = \min(T_1 - (T_3 - b), \frac{1}{2}c)$, so after shifting by k, the clocks of p_1 and p_3 differ by at most $\max(T_1 - T_3 + b + k, \frac{1}{2}c - k)$ (the second term comes from the possibility that p_3's clock read T earlier than that of p_1). Thus, p_2 should choose k to minimize

$$F(k) = \max(\tfrac{1}{2}(T_3 - k - T_1 + a), T_1 - T_3 + b + k, \tfrac{1}{2}c - k),$$

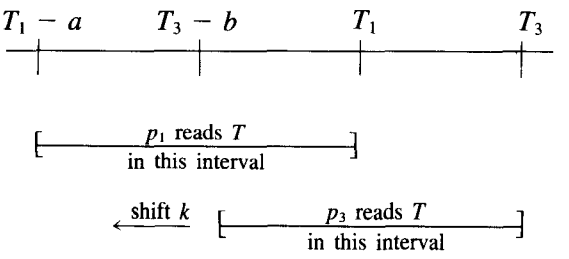

FIG. 2.

subject to the constraint that $0 \leq k \leq \min(T_3 - b - T_1 + a, T_3 - T_1)$. It is now easy to check that k can always be chosen so that $F(k) \leq \max(\frac{1}{3}(a + b), \frac{1}{2}c)$. Indeed, if we set $k = T_3 - T_1 + \frac{1}{3}(a - 2b)$ (provided this satisfies the constraints), then $F(k) = \max(\frac{1}{3}(a + b), \frac{1}{2}c - k)$. If this choice of k does not satisfy the constraints, then we leave it to the reader to check that setting k to one of 0 or $\min(T_3 - b - T_1 + a, T_3 - T_1)$ will work. Since $\Delta_{C_3} \leq \min_k(F(k))$, we are done. ■

Proposition 3.4 already suggests some of the difficulties involved in giving an algorithm for attaining Δ_C for an arbitrary communication network C. It also illustrates two of the main ideas used in our optimal algorithm: comparing the time on a time-stamped message with local time, and choosing a shift that will minimize a certain expression. These ideas are explored in more detail in the next section.

4. An Algorithm for Attaining Optimal Precision

We now develop a general framework for solving the problem of achieving optimal precision. As mentioned in Section 2, it suffices that each processor be able to compute a value for its local variable TAR in such a way as to synchronize logical clocks as tightly as possible. Thus the processors' aim is to estimate as closely as possible the difference in readings between their physical clocks.

We can view the process as a game, where "nature" plays against the processors. The game has three phases which can roughly be described as follows. In the first phase, nature chooses the readings for each processor's physical clock (since we assume that processors' physical clocks run at the rate of real time, it suffices for nature to choose an initial setting $D_i(0)$ for each processor p_i), and a message transmission time for each message, subject to the constraint that messages from p_i to p_j must take between $L(p_i, p_j)$ and $H(p_i, p_j)$. The processors do not know the choices made by nature, but they can observe the difference between the time a message is sent (on the sending processor's physical clock) and the time it is received (on the receiving processor's physical clock). Suppose p_i sends a message to p_j, time stamped T, which is received by p_j at time T' on its (p_j's) clock. Let δ_{ij} be the actual transmission time of this message. Then it is easy to see that $T' - T = D_j(0) - D_i(0) + \delta_{ij}$. Since $L(p_i, p_j) \leq \delta_{ij} \leq H(p_i, p_j)$, it follows that

$$L(p_i, p_j) - (T' - T) \leq D_i(0) - D_j(0) \leq H(p_i, p_j) - (T' - T).$$

Thus, in phase 2, processors send messages to each other and then use the information acquired by this message exchange to try to compute the shifts that will bring their logical clocks optimally close together. In phase 3, nature can revise its choices of initial physical clock readings so as to maximize the largest difference between the clock readings of processors; i.e., if SHIFT_i is

the final value of p_i's variable TAR as chosen in the second phase, nature wants to choose INIT_i, $i = 1, \ldots, n$, initial readings for the processors' clocks, so as to maximize:

$$\max_i(\text{INIT}_i + \text{SHIFT}_i) - \min_j(\text{INIT}_j + \text{SHIFT}_j).$$

Of course, nature is constrained to choose the values of the INIT_i's in a way that is consistent with the information received by the processors (i.e., so that the resulting run is indistinguishable from that initially chosen).

Keeping these ideas in mind, consider the following "pseudoalgorithm" for achieving optimal precision in a network $C = (G, H, L)$ consisting of processors $p_1, \ldots, p_n$. (We call it a pseudoalgorithm since it is not yet clear how to implement step 4.)

1. Each processor sends all its neighbors a message time stamped with its current (physical) clock reading.
2. If p_i receives a message from a neighbor p_j time stamped T at time T' on its (physical) clock, p_i computes $\tau_{ij} = T' - T$.
3. The values τ_{ij} are all sent to one processor, say p_1.
4. Processor p_1 computes $\text{SHIFT}_1, \ldots, \text{SHIFT}_n$ so as to

Minimize

$$\max_{\text{INIT}_1, \ldots, \text{INIT}_n}(\max_i(\text{INIT}_i + \text{SHIFT}_i) - \min_j(\text{INIT}_j + \text{SHIFT}_j)) \quad (*)$$

$$\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}.$$

5. p_1 tells p_j the value SHIFT_j computed in step 4; p_j then sets $\text{TAR}_j := \text{SHIFT}_j$.

We now show that this pseudoalgorithm is optimal; we then show how to implement it in polynomial time. In fact, the pseudoalgorithm is optimal in a strong sense: it gives the tightest precision possible over all runs consistent with nature's choice of initial clock readings and message transmission times. Intuitively this is because the pseudoalgorithm makes optimal use of all the information present in the system.

To make this precise, we first need some definitions. We define a run r to be *standard* if for all i, j, there is a constant δ_{ij} such that all messages from p_j to p_j take time δ_{ij}. Given a run r, let $\text{INIT}_i(r)$ be the initial reading of p_i's physical clock in run r, and let $\delta_{ij}(r)$ be the message transmission time of the first message from p_i to p_j if there is such a message (otherwise $\delta_{ij}(r)$ is undefined). Given a communication network C, two runs r and r' in C are said to be *consistent* if for all i,j whenever $\delta_{ij}(r)$ and $\delta_{ij}(r')$ are both defined, we have $\text{INIT}_j(r) - \text{INIT}_i(r) + \delta_{ij}(r) = \text{INIT}_j(r') - \text{INIT}_i(r') + \delta_{ij}(r')$. In case there is no first message from p_i to p_j in run r, but there is one in run r', we add the technical condition that $\text{INIT}_j(r) - \text{INIT}_i(r) \leq \text{INIT}_j(r') - \text{INIT}_i(r') + \delta_{ij}(r')$; there is a symmetric condition in case $\delta_{ij}(r)$ is defined but

$\delta_{ij}(r')$ is not. As we have already observed, if r is a run of an algorithm that implements the pseudoalgorithm described above, the quantity τ_{ij} computed in step 2 is equal to $\mathrm{INIT}_j(r) - \mathrm{INIT}_i(r) + \delta_{ij}(r)$. Thus, in any implementation of the pseudoalgorithm, the same shifts are computed for all consistent runs.

The following lemma generalizes some of the key ideas of the proof of the lower bound of Proposition 3.1.

LEMMA 4.1. *Let C be a communication network and $\mathcal{A}$ be any algorithm. If r and r' are two consistent standard runs of $\mathcal{A}$, then r and r' are indistinguishable in that all processors receive the same messages at the same time on their physical clocks in both runs. In particular, $\Delta_{C,\mathcal{A}}(r) = \Delta_{C,\mathcal{A}}(r')$.*

Proof. Suppose not. We then derive a contradiction by considering the first time that r and r' differ. More precisely, let T_i be the first time on p_i's physical clock that p_i's message history differs in r and r'; we take $T_i = \infty$ if p_i has the same message history in both r and r'. (Note that if p_i's message history differs in r and r', there will be a first message that causes a discrepancy, by our assumption that there are only finitely many messages in a processor's message history.) If $T_i < \infty$, let t_i be the real time such that $D_i(t_i) = T_i$ in run r. Suppose j is such that t_j is minimal. Thus t_j is the first real time (as measured in run r) where there is a discrepancy in the message history of some processor. There must be a tuple $\langle p_k, m, T_j, y\rangle$ in p_j's message history in run r which is not in p_j's message history in run r', or there is such a tuple in p_j's message history in r' which is not in p_j's message history in r. This tuple cannot represent a message that was sent by p_j, since the messages sent only depend on the prior message history, which is the same for p_j in both runs. Thus, this must be a message that was received from p_k. Suppose the message was received in r, but not in r', and suppose p_k sent the message m to p_j at time T' on its physical clock in run r. There are now two subcases to consider: (a) p_k also sends the message m to p_j at time T' on its physical clock in run r', or (b) it does not. Suppose subcase (a) holds, and p_j receives the message m at time U in run r'. Since r is a standard run, all messages from p_k to p_j take $\delta_{kj}(r)$, so we easily get $T_j - T' = \mathrm{INIT}_j(r) - \mathrm{INIT}_k(r) + \delta_{kj}(r)$. Similarly, $U - T' = \mathrm{INIT}_j(r') - \mathrm{INIT}_k(r') + \delta_{kj}(r')$. Since r and r' are consistent, we immediately get $U - T' = T_j - T'$. Thus $U = T_j$, contradicting the assumption that there is a discrepancy between p_j's message histories in runs r and r' at time T_j.

Now consider subcase (b). Let t' be the real time that p_k's physical clock reads T' in run r (thus t' is the real time that p_k sends the message m to p_j in run r). Note $t_j - t' = \delta_{kj}(r)$, so $t' \leq t_j$. Since p_k does not send the message in run r' when its clock reads T', there must be a discrepancy in the message histories of p_k in the two runs *before* p_k's physical clock reads T' (since the messages that p_k sends at clock time T' depend on message history up to, but not including, T'). Thus $t_k < t' \leq t_j$, contradicting the choice of t_j.

Finally, suppose $\langle p_k, m, T_j \rangle$ is a tuple in p_j's message history in run r but not in r'. Again there are two subcases. The argument in the first subcase is identical to that above, while for subcase (b), if again we take T' to be the time on p_k's physical clock in run r' when it sent the message m to p_j, from the fact that r and r' are consistent, we get that $t_j - t' = \delta_{kj}(r)$, if $\delta_{kj}(r)$ is defined. If not, it is easy to check that the technical condition added to the definition of consistency ensures that we still have $t_j - t' = (T_j - \mathrm{INIT}_j(r)) - (T' - \mathrm{INIT}_k(r)) = (\mathrm{INIT}_j(r') - \mathrm{INIT}_k(r') + \delta_{kj}(r')) - (\mathrm{INIT}_j(r) - \mathrm{INIT}_k(r)) \geq 0$. Now we can complete the argument just as above. ■

THEOREM 4.2. *Let C be a communication network and $\mathcal{B}$ be an implementation of the pseudoalgorithm above. Then $\Delta_{C,\mathcal{B}} = \Delta_C$. Moreover, if r is a run of $\mathcal{B}$, and $\mathcal{A}$ is any other algorithm, then*

$$\Delta_{C,\mathcal{B}}(r) \leq \max_s\{\Delta_{C,\mathcal{A}}(s) \mid s \text{ is consistent with } r\}. \qquad (\dagger)$$

Proof. It suffices to prove (†). For then we have, for any algorithm $\mathcal{A}$,

$$\begin{aligned}
\Delta_{C,\mathcal{B}} &= \max_r \Delta_{C,\mathcal{B}}(r) \\
&\leq \max_r(\max_s\{\Delta_{C,\mathcal{A}}(s) \mid s \text{ is consistent with } r\}) \qquad \text{(by (†))} \\
&\leq \max_s \Delta_{C,\mathcal{A}}(s) \\
&= \Delta_{C,\mathcal{A}}.
\end{aligned}$$

Thus $\Delta_C = \min_{\mathcal{A}}\{\Delta_{C,\mathcal{A}}\} = \Delta_{C,\mathcal{B}}$.

In order to prove (†), let r be a run of $\mathcal{B}$, and suppose the values τ_{ij} are the ones computed in step 2 of $\mathcal{B}$ in run r. Suppose $\mathrm{INIT}_1, \ldots, \mathrm{INIT}_n$ satisfy the constraint of (*). Let s be a standard run of $\mathcal{A}$ such that $\mathrm{INIT}_i(s) = \mathrm{INIT}_i$, $i = 1, \ldots, n$, and $\delta_{ij}(s) = \mathrm{INIT}_j - \mathrm{INIT}_i + \tau_{ij}$ (provided there are messages from p_i to p_j in s). By the constraint of (*), it follows that $L(p_i, p_j) \leq \delta_{ij}(s) \leq H(p_i, p_j)$. It is also easy to see that if $\delta_{ij}(s)$ is defined, then $\mathrm{INIT}_j(r) - \mathrm{INIT}_i(r) + \delta_{ij}(r) = \tau_{ij} = \mathrm{INIT}_j(s) - \mathrm{INIT}_i(s) + \delta_{ij}(s)$. If $\delta_{ij}(s)$ is not defined, it is also easy to check that the technical condition for consistency holds. Thus r and s are consistent.

Thus, for any choice of $\mathrm{INIT}_1, \ldots, \mathrm{INIT}_n$ satisfying the constraint of (*), there is a standard run s of $\mathcal{A}$ consistent with r such that $\mathrm{INIT}_i(s) = \mathrm{INIT}_i$, $i = 1, \ldots, n$. Let S be the set of all standard runs of $\mathcal{A}$ that are consistent with r. Note that by Lemma 4.1, all the runs in S are indistinguishable. Let $\mathrm{SHIFT}_1, \ldots, \mathrm{SHIFT}_n$ be the shifts computed by $\mathcal{A}$ in any one run, and hence all runs, in S. Thus we have

$$\begin{aligned}
&\max_s\{\Delta_{C,\mathcal{A}}(s) \mid s \text{ is a consistent with } r\} \\
&\geq \max_{s \in S} \Delta_{C,\mathcal{A}}(s)
\end{aligned}$$

$$\geq \max_{\text{INIT}_1, \ldots, \text{INIT}_n}\{\max_i(\text{INIT}_i + \text{SHIFT}_i) - \min_j(\text{INIT}_j + \text{SHIFT}_j) \mid$$
$$L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}\}$$
$$\geq \Delta_{C,\mathcal{B}}(r).$$

The result is now immediate. ■

We remark that the proof here suggests that nature's best strategy is to use standard runs. The intuition behind this is that the least amount of information is released in standard runs. Varying the transmission time of messages between p_i and p_j allows the processors to gain extra information about their relative initial time readings, by repeated measurements of τ_{ij}. Thus in our algorithm we only measure the value of τ_{ij} once, on the assumption that nature uses its best strategy.

It now remains to show how to implement step 4 in the pseudoalgorithm. For this we need some techniques of linear programming and combinatorial optimization.

First consider "nature's problem" in Phase 3 of the game, namely, choosing $\text{INIT}_1, \ldots, \text{INIT}_n$, given $\text{SHIFT}_1, \ldots, \text{SHIFT}_n$ and $\{\tau_{ij} \mid p_i \text{ and } p_j \text{ are neighbors}\}$ so as to

$$\begin{aligned}&\text{Maximize } (\max_i(\text{INIT}_i + \text{SHIFT}_i) - \min_j(\text{INIT}_j + \text{SHIFT}_j))\\ &\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}.\end{aligned} \qquad (**)$$

This problem can be solved as follows. For every pair of nodes (p_k, p_l), let D_{kl} denote the maximum of $\text{INIT}_k - \text{INIT}_l$ that can be obtained by choosing INIT_i's subject to the above constraints. Thus D_{kl} is obtained by choosing $\text{INIT}_1, \ldots, \text{INIT}_n$ so as to

$$\begin{aligned}&\text{Maximize } \text{INIT}_k - \text{INIT}_l\\ &\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}.\end{aligned}$$

Since the constraint $L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}$ is equivalent to $\tau_{ji} - H(p_j, p_i) \leq \text{INIT}_j - \text{INIT}_i \leq \tau_{ji} - L(p_j, p_i)$ we can reformulate the preceding problem to choosing $\text{INIT}_1, \ldots, \text{INIT}_n$ so as to

$$\begin{aligned}&\text{Maximize } \text{INIT}_k - \text{INIT}_l\\ &\text{subject to } \text{INIT}_i - \text{INIT}_j \leq \min(H(p_i, p_j) - \tau_{ij}, \tau_{ji} - L(p_j, p_i)).\end{aligned}$$

The latter problem is the dual of the shortest path problem from p_k to p_l, relative to the distances $d_{ij} = \min(H(p_i, p_j) - \tau_{ij}, \tau_{ji} - L(p_j, p_i))$ (see, for example, Lawler (1976) for a general discussion of duality and problems of this nature). Thus, D_{kl} is the length of the shortest path from p_k to p_l relative to the d_{ij}'s.

Now, consider the processors' problem: solving (*) given the τ_{ij}'s. In the worst case the maximum difference between the logical clocks of any two processors will be equal to the maximum over all pairs (p_k, p_l) of the quantity $D_{kl} + \text{SHIFT}_k - \text{SHIFT}_l$. Thus (*) reduces to finding $\text{SHIFT}_1, \ldots, \text{SHIFT}_n$ so as to

$$\text{Minimize } \max_{k,l}(D_{kl} + \text{SHIFT}_k - \text{SHIFT}_l).$$

This is equivalent to finding $\text{SHIFT}_1, \ldots, \text{SHIFT}_n$ so as to

$$\text{Minimize } \lambda$$

$$\text{subject to } \text{SHIFT}_k - \text{SHIFT}_l \leq \lambda - D_{kl}.$$

We now provide a characterization of the solution of this problem. Consider any set of values for λ and the SHIFT_i's. Let $i_1, \ldots, i_m$ be a circuit (i.e., $i_1 = i_m$). If we take the sum of inequalities associated with edges around this circuit, then the left-hand side $(\sum_{j=1}^{m-1}(\text{SHIFT}_{j+1} - \text{SHIFT}_j))$ is identically zero and hence the total weight of $\lambda - D_{kl}$ around the circuit is nonnegative. Let λ^* denote the minimum value of λ. Consider any circuit K. If K has m edges then λ^* is greater than or equal to the total (D_{kl})-weight around K, divided by m. We have thus established that λ^* is greater than or equal to the maximum (over all directed circuits) of the average (D_{kl})-weight per edge around the circuit. We now use a duality argument to show that λ^* actually equals the maximum average weight per edge along a circuit. The dual linear programming problem that determines λ^* is the following:

$$\text{Maximize } \sum_{ij} D_{ij}x_{ij}$$

$$\text{subject to } \sum_{j}(x_{ij} - x_{ji}) = 0 \qquad \text{for all } i$$

$$\sum_{ij} x_{ij} = 1$$

$$x_{ij} \geq 0.$$

A matrix (x_{ij}) satisfying the first and third constraints above $(\Sigma_j(x_{ij} - x_{ji}) = 0$ and $x_{ij} \geq 0)$ is usually called in the literature a *circulation*. If in addition it satisfies the second constraint $(\Sigma_{ij}x_{ij} = 1)$ then we call it a *normal circulation*. If we interpret x_{ij} as the *flow* in the link from i to j, then the first constraint implies conservation of flow at each node. It is well known that any circulation can be represented as a sum of *simple circuit circulations* (cf. Lawler, 1976), where a simple circuit circulation is one in which the links ij for which $x_{ij} > 0$ form a simple circuit. Thus a normal circuit circulation can be represented as a convex combination of normal simple circuit circulations.

Note that from the first constraint, it follows that all nonzero flows in a simple circuit circulation must be equal, so in a normal simple circuit circulation where the circuit has length m, the value of the x_{ij} around the circuit must be $1/m$. It easily follows that the circulation that maximizes the optimization problem is one such normal simple circuit circulation, hence the value of λ^* is the average (D_{kl})-weight over some circuit.

The value of λ^* as well as a corresponding circuit can be computed in $O(n^3)$ time (where n is the number of processors), using an algorithm of Karp (1978). It follows that around that circuit the original inequalities are satisfied as equalities: $\text{SHIFT}_k - \text{SHIFT}_l = \lambda^* - D_{kl}$. Thus, the SHIFT_i's can be easily computed around the circuit by setting one of them arbitrarily to zero and computing the rest using the above equalities. All the other SHIFT_i's can be produced by a shortest path computation starting from the node p_i for which $\text{SHIFT}_i = 0$, using the distances $\lambda^* - D_{kl}$.

The discussion above leads to the following theorem.

THEOREM 4.3. *Let $C = (G, H, L)$, where $G = \{p_1, \ldots, p_n\}$. Let D_{ij} denote the length of the shortest path from p_i to p_j relative to the edge weights $d_{ij} = \min(H(p_i, p_j) - \tau_{ij}, \tau_{ji} - L(p_j, p_i))$. Let Δ denote the maximum circuit mean relative to the edge weights D_{ij}. Then Δ is the solution to (*). Moreover, the values of $\text{SHIFT}_1, \ldots, \text{SHIFT}_n$ that give Δ can be computed in $O(n^3)$ time.*

Putting together Theorems 4.2 and 4.3, we see that we have a polynomial time algorithm that guarantees optimal precision in any communication network. To actually compute the value of Δ_C, we need to compute nature's optimal strategy in phase 1, which is to choose τ_{ij}'s in order to

$$\text{Maximize } \min_{\text{SHIFT}_1, \ldots, \text{SHIFT}_n} (\max_{\text{INIT}_1, \ldots, \text{INIT}_n} (\max_i(\text{INIT}_i + \text{SHIFT}_i) - \min_j(\text{INIT}_j + \text{SHIFT}_j))) \qquad (\ddagger)$$

$$\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}$$

(We remark that we have worded this problem in terms of choosing τ_{ij}'s, which are not directly observable by nature. However, since $\tau_{ij} = D_j(0) - D_i(0) + \delta_{ij}$, we can easily reformulate this problem in terms of variables under nature's control. Indeed, it is easy to see that the solution to the maximum circuit mean problem is invariant under transformations of the form $d'_{ij} = d_{ij} + \pi_i - \pi_j$, and from this fact it is not hard to show that replacing τ_{ij} by δ_{ij} in (‡) or (*) results in an *equivalent* problem.)

By the discussion above, this amounts to choosing the τ_{ij}'s in order to maximize the value of λ^*, the maximum circuit mean (i.e. the maximum mean D_{kl}-weight over all simple circuits). This problem can be solved in

exponential time. To see this, note that there are less than $(n + 1)!$ circuits in the graph, where n is the number of nodes (i.e., the number of processors). Thus it suffices to show that given a circuit, the subproblem of computing τ_{ij}'s so as to maximize the mean D_{kl}-weight around that circuit can be solved in exponential time. We can clearly maximize the mean D_{kl}-weight around a circuit by maximizing the sum of the D_{kl}'s around the circuit. But recall that D_{kl} is obtained by choosing $\text{INIT}_1, \ldots, \text{INIT}_n$ so as to

$$\text{Maximize } \text{INIT}_k - \text{INIT}_l$$
$$\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i - \text{INIT}_j \leq H(p_i, p_j) - \tau_{ij}.$$

Suppose without loss of generality we are given the circuit $(1, 2, \ldots, m, 1)$, where $2 \leq m \leq n$ (so that node $m + 1$ is identified with 1). Thus the maximum D_{kl} sum around the circuit can be found by solving the following linear programming problem: Choose INIT_i^k, $i = 1, \ldots, n$, $k = 1, \ldots, m$, and τ_{ij} so as to

$$\text{Maximize } \sum_{k=1}^{m} \text{INIT}_{k+1}^k - \text{INIT}_k^k$$
$$\text{subject to } L(p_i, p_j) - \tau_{ij} \leq \text{INIT}_i^k - \text{INIT}_j^k \leq H(p_i, p_j) - \tau_{ij},$$
$$k = 1, \ldots, m, \qquad i, j = 1, \ldots, n.$$

Of course, this linear programming problem can be solved in polynomial time using well-known techniques, but to prove that our problem can be solved in exponential time all we need is that the linear programming problem can be solved in exponential time.

Since the discussion prior to Theorem 4.3 shows that Δ_C is in fact the maximum circuit mean over all choices of τ_{ij}, in the process of solving nature's problem, we also compute the value of Δ_C. Thus we have:

THEOREM 4.4. *Let $C = (G, H, L)$, where $G = \{p_1, \ldots, p_n\}$. Then a solution to (‡) and the value of Δ_C can be found in time $O(2^{cn\log(n)})$, for some constant $c > 0$.*

It also follows from the discussion above that the following problem is in NP: given a communication network C with rational data and a rational number Δ, decide if $\Delta_C \geq \Delta$. To solve this problem, we simply have to guess a circuit and check that the mean over that circuit can be made greater than or equal to Δ by appropriately choosing the τ_{ij}'s. Note that here we rely only on the fact that the linear programming problem is in NP and not on the existence of polynomial-time algorithms for solving it. We conjecture that in fact our problem (‡) is NP complete, but have not yet proved this.

5. Probabilistic Algorithms

When decisions have to be made under uncertainty one can usually do better by randomizing. In the context of our problem, the question is whether the processors can do any better by using a probabilistic algorithm for choosing their shifts, given all the information they have collected. In fact, we show that they cannot.

We sketch the semantics of probabilistic algorithms here, without going into details. An *augmented run* of a probabilistic algorithm $\mathcal{A}$ is a pair (r, c), where r is a run of $\mathcal{A}$ (i.e., a specification of physical clock readings for each processor, and message transmission times for each message) and c is a sequence of outcomes of coin tosses for each processor. Note that the behavior of a processor is completely specified in an augmented run of a probabilistic algorithm $\mathcal{A}$.

By analogy to the definitions of Section 2, we define $\Delta_{C,\mathcal{A}}(p, p', r, c)$ to be the difference between the real times when p and p' change the value of their special register, in the augmented run of probabilistic algorithm $\mathcal{A}$. We take $\Delta_{C,\mathcal{A}}(r, c) = \max_{p,p'}\{\Delta_{C,\mathcal{A}}(p, p', r, c)\}$ and $\Delta_{C,\mathcal{A}}(r)$ to be the expected value of $\Delta_{C,A}(r, c)$ over all coin tosses c (note that this generalizes the definition given for deterministic $\mathcal{A}$, where we can view the algorithm's behavior as independent of c). We can then define $\Delta_{C,\mathcal{A}} = \max_r \Delta_{C,\mathcal{A}}(r)$, just as before.

We will show that $\Delta_{C,\mathcal{A}} \geq \Delta_{C,\mathcal{B}}$, where $\mathcal{B}$ is an implementation of the optimal algorithm of Section 4. Thus, the mean imprecision inherent in a probabilistic algorithm is at least as great as that of the optimal deterministic algorithm, in fact, we actually prove a stronger result.

Given a communication network C and an algorithm $\mathcal{A}$, there is *some* run of $\mathcal{A}$ where the maximum difference between when two processors change the value of their special register is at least Δ_C. But this run could a priori depend in a complicated way on $\mathcal{A}$. We now show that there is a set of m scenarios, say $r_1, \ldots, r_m$ (where m is the number of nodes in the circuit with the maximum circuit mean constructed in the proof of Theorem 4.3), such that for any (deterministic or probabilistic) algorithm $\mathcal{A}$, the average value of $\Delta_{C,\mathcal{A}}(r_i)$, $i = 1, \ldots, m$, is at least Δ_C, and hence $\Delta_{C,\mathcal{A}}(r_i) \geq \Delta_C$ for some i. (We remark that for the optimal algorithm $\mathcal{B}$, it turns out that $\Delta_{C,\mathcal{B}}(r_i) = \Delta_C$ for all these m scenarios; in fact, one way of looking at $\mathcal{B}$ is that it is defined to have precisely this property.) Note that once we have proved this claim, it will follow that nature has a probabilistic strategy that "neutralizes" *any* algorithm the processors may use. It simply chooses one of these scenarios at random, with equal probability. In terms of the three-phase game discussed at the beginning of Section 4, this means that nature can pass up its move at the third phase just by using this randomized strategy at phase 1.

To understand the intuition behind this, let us reconsider the two-processor network C discussed in Proposition 3.1, where the uncertainty for messages

between p and q is a. Recall the two scenarios described in the proof of Proposition 3.1: (1) $D_p(0) = D_q(0)$, $\delta_{pq} = a$, $\delta_{qp} = 0$; and (2) $D_q(0) = D_p(0) + a$, $\delta_{pq} = 0$, $\delta_{qp} = a$. For any deterministic algorithm $\mathcal{A}$, the runs corresponding to these scenarios, call them r_1 and r_2, will be indistinguishable. It is also easy to extend the arguments of Proposition 3.1 to show that $\frac{1}{2}(\Delta_{C,\mathcal{A}}(r_1) + \Delta_{C,\mathcal{A}}(r_2)) \geq \frac{1}{2}a$; i.e., the average precision over these two runs is at least the optimal precision. For a probabilistic algorithm $\mathcal{A}$, an analogous argument shows that for any sequence c of coin tosses, the augmented runs (r_1, c) and (r_2, c) are indistinguishable, so that $\frac{1}{2}(\Delta_{C,\mathcal{A}}(r_1, c) + \Delta_{C,\mathcal{A}}(r_2, c)) \geq \frac{1}{2}a$. Taking the expected value over all sequences of coin tosses, we still get $\frac{1}{2}(\Delta_{C,\mathcal{A}}(r_1) + \Delta_{C,\mathcal{A}}(r_2)) \geq \frac{1}{2}a$.

In a general network C, nature's first step is to choose τ_{ij}'s to satisfy Eq. (‡). With this choice of τ_{ij}'s, nature then computes the D_{ij}'s used to solve Eq. (*). Consider the values of D_{ij} for which the maximum circuit mean is attained. For ease of notation, we assume without loss of generality that the maximum circuit mean is attained at the circuit $(1, 2, \ldots, m, 1)$ (for some $2 \leq m \leq n$). Let r_k, $k = 1, \ldots, m$, be the m scenarios defined via $\mathrm{INIT}_i(r_k) = -D_{ki}$ and all messages from p_i to p_j in r_k have transmission time $\tau_{ij} + \mathrm{INIT}_i(r_k) - \mathrm{INIT}_j(r_k)$. Of course, we must show that this choice of message transmission times satisfies the constraints in the network. By construction, we have $\mathrm{INIT}_i(r_k) - \mathrm{INIT}_j(r_k) = D_{kj} - D_{ki}$. Since D_{ij} is the length of the shortest path from p_i to p_j relative to the distances $d_{ij} = \min(H(p_i, p_j) - \tau_{ij}, \tau_{ji} - L(p_i, p_j))$, it is easy to see that we have $D_{kj} - D_{ki} \leq d_{ij}$. Thus it follows that $L(p_i, p_j) - \tau_{ij} \leq \mathrm{INIT}_i(r_k) - \mathrm{INIT}_j(r_k) \leq H(p_i, p_j) - \tau_{ij}$. Thus, this choice of message transmission times does satisfy the constraints of the network.

Note that by construction, for any algorithm $\mathcal{A}$, the scenarios will be consistent. Hence, if $\mathcal{A}$ is a deterministic algorithm, the same values $\mathrm{SHIFT}_1, \ldots, \mathrm{SHIFT}_n$ will be chosen in all these runs. Let l denote the successor of k in the maximum circuit. Since $\mathrm{INIT}_k(r_k) - \mathrm{INIT}_l(r_k) = D_{kl}$, the maximum difference between final logical clock readings in r_k is at least $D_{kl} + \mathrm{SHIFT}_k - \mathrm{SHIFT}_l$. Therefore the average value of the maximum difference between the final readings of the processors' logical clocks over these m scenarios is at least the average of the $D_{kl} + \mathrm{SHIFT}_k - \mathrm{SHIFT}_l$, where k varies around the circuit. The latter however is precisely the mean of the D_{kl}'s around the circuit, which by the arguments of Section 4, is just Δ_C.

If $\mathcal{A}$ is a probabilistic algorithm, a similar argument works. Let c be any sequence of coin tosses. An argument identical to that used in Lemma 4.1 shows that all the augmented runs $(r_1, c), \ldots, (r_m, c)$ are indistinguishable. Again, let $\mathrm{SHIFT}_1, \ldots, \mathrm{SHIFT}_n$ be the shifts computed in one, and hence in all, of these augmented runs. The same argument as that used above shows that the average value of the maximum difference in these m augmented runs is at least Δ_C. Taking the expected value over all sequences of coin tosses c, we get the desired result.

The previous discussion can be summarized as follows:

THEOREM 5.1. *Let C be a communication network. There are m scenarios $r_1, \ldots, r_m$ in C (where m is the size of the maximum circuit mean of Theorem 4.3) such that for any algorithm $\mathcal{A}$, $(1/m)(\sum_{i=1}^{m} \Delta_{C,\mathcal{A}}(r_i)) \geq \Delta_C$ (and thus $\Delta_{C,\mathcal{A}}(r_i) \geq \Delta_C$ for some i).*

6. IMPRECISION IN UNDIRECTED NETWORKS

In this section we show that we can always reduce the problem of calculating the imprecision in a network to that of calculating the imprecision of a completely connected, undirected network, where the lower bound on message transmission time is 0 over every link. For an undirected network, we take the H function to be symmetric, so that $H(p, q) = H(q, p)$.

We prove this result in stages, making heavy use of Theorem 4.2, which says that optimal precision is attained by solving the combinatorial optimization problem of Eq. (*). For this section, we always use $\mathcal{B}$ to denote an implementation of the optimal algorithm for attaining precision, as described in Section 4. The following easy lemma gives us a way to compare imprecision in two networks:

LEMMA 6.1. *Let C and C' be two communication networks. If for all runs r of $\mathcal{B}$ in C, there is a run r' of $\mathcal{B}$ in C' such that $\Delta_{C,\mathcal{B}}(r) \leq \Delta_{C',\mathcal{B}}(r')$, then $\Delta_C \leq \Delta_{C'}$.*

Proof. Suppose the hypotheses of the lemma hold. Then since $\mathcal{B}$ is an optimal algorithm, we have $\Delta_C = \Delta_{C,\mathcal{B}} = \max_r(\Delta_{C,\mathcal{B}}(r)) \leq \max_{r'}(\Delta_{C',\mathcal{B}}(r')) = \Delta_{C',\mathcal{B}} = \Delta_{C'}$. ■

Our first proposition shows that the imprecision in a network depends only on the uncertainty in message transmission time.

PROPOSITION 6.2. *Let $C = (G, H, L)$ be a communication network, with $G = (V, E)$, and $C' = (G, H', L')$, where for all edges $(p, q) \in E$, we have $L'(p, q) = 0$ and $H'(p, q) = H(p, q) - L(p, q)$. Then $\Delta_C = \Delta_{C'}$.*

Proof. Given a run r in C, define a run r' in C' via $\mathrm{INIT}_i(r') = \mathrm{INIT}_i(r)$ and $\delta_{ij}(r') = \delta_{ij}(r) - L(p_i, p_j)$. By construction, $0 \leq \delta_{ij}(r') \leq H(p_i, p_j) - L(p_i, p_j)$, so r' is a scenario in C'. Let τ_{ij} (resp. τ'_{ij}) be the quantities computed in step 2 of $\mathcal{B}$ in run r (resp. r'). Thus, the constraints on the optimization problem (*) in r' are $-\tau'_{ij} \leq \mathrm{INIT}_i - \mathrm{INIT}_j \leq H'(p_i, p_j) - \tau'_{ij}$, while the constraints in r are $L(p_i, p_j) - \tau_{ij} \leq \mathrm{INIT}_i - \mathrm{INIT}_j \leq H(p_i, p_j)$. Since we have $\tau_{ij} = \mathrm{INIT}_j(r) - \mathrm{INIT}_i(r) + \delta_{ij}(r)$, $\tau'_{ij} = \mathrm{INIT}_j(r') - \mathrm{INIT}_i(r') + \delta_{ij}(r')$, $\mathrm{INIT}_i(r) = \mathrm{INIT}_i(r')$, and $\mathrm{INIT}_j(r) = \mathrm{INIT}_j(r')$, it is easy to see that we have $\tau'_{ij} = \tau_{ij} - L(p_i, p_j)$. And by construction, $H'(p_i, p_j) = H(p_i, p_j) - L(p_i, p_j)$. Now an easy computation shows that in fact the constraints in runs r and r' are identical.

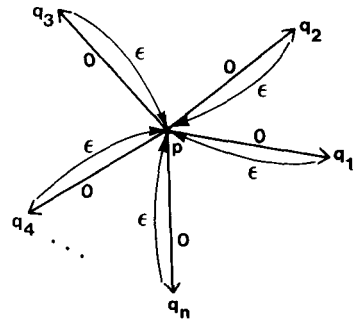

FIG. 3.

Since the constraints in r and r' are identical, the solutions to Eq. (*) must be identical in r and r', hence $\Delta_{C,\mathcal{B}}(r) = \Delta_{C',\mathcal{B}}(r')$. Thus by Lemma 6.1, we have $\Delta_C \leq \Delta_{C'}$. The opposite inequality is proved by a symmetric argument. We omit details here. ■

We next show that to every directed network, there corresponds an undirected network where the H function is symmetric. To get some intuition for this result, we examine two simple examples. First consider the star-shaped network of Fig. 3, with central node p and outlying nodes $q_1, \ldots, q_n$. Suppose we have $L(p, q_i) = L(q_i, p) = 0$, $H(p, q_i) = 0$, and $H(q_i, p) = \epsilon$, for $i = 1, \ldots, n$ and $\epsilon > 0$. It is easy to see that the imprecision is 0 in this network: p simply broadcasts a time T to all the other processors, and they set their logical clocks to T upon receipt of this message. Since message transmission time is 0, they must be completely synchronized.

Now suppose we flip the uncertainties, taking $H(p, q_i) = \epsilon$ and $H(q_i, p) = 0$ (see Fig. 4). Again the imprecision is 0! Each of the q_i's sends a message to the central node p with its local clock reading. Upon receipt of these messages, p can easily compute the amount by which each q_i must shift

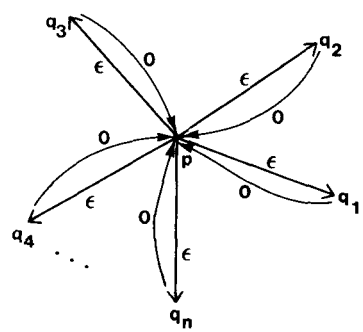

FIG. 4.

its clock so that all logical clocks are synchronized to that of p. Thus p computes these shifts and sends them out to each processor. Note that although there is some uncertainty about when the message from p reporting the shift will arrive, complete synchronization can still be attained.

We can summarize the intuition behind these two examples as follows. If the uncertainty in the link from p to q is less than the uncertainty in the link from q to p, then p sends q a time-stamped message and q adjusts its logical clock accordingly. If the uncertainty is less in the link from q to p, then q sends p a time-stamped message, p computes a shift, and sends it back to q, which again adjusts its logical clock accordingly. In both cases, the imprecision in the network is the same as that of a new network where the uncertainty in the link from p to q is the minimum of the uncertainties in the link from p to q and the link from q to p in the original network. These ideas can be generalized as follows.

Given a communication network $C = (G, H, L)$ with $G = (V, E)$ and $L(p, q) = 0$ for all edges $(p, q) \in E$, let the communication network $C_{\text{sym}} = (G', H', L')$, where $G' = (V, E')$, where $E' = \{(p, q) \mid (p, q) \in E \text{ or } (q, p) \in E\}$, $L'(p, q) = 0$ and $H'(p, q) = \min(H(p, q), H(q, p))$ for all nodes $(p, q) \in E'$ (where we take $H(p, q) = \infty$ if $(p, q) \notin E$). Note that H' is symmetric.

PROPOSITION 6.3. $\Delta_{C_{\text{sym}}} = \Delta_C$.

Proof. It is easy to see that $\Delta_{C_{\text{sym}}} \leq \Delta_C$, since there are more constraints on the solutions to (*) for C_{sym} due to the tighter bounds on message transmission time. More formally, given a run r of $\mathcal{B}$ in C_{sym}, let r' be the corresponding run of $\mathcal{B}$ in C, where the same choices are made for physical clock readings and message transmission times (note that r' is indeed a run in C since any legal message transmission time in C_{sym} is also a legal message transmission time in C). Clearly $\Delta_{C,\mathcal{B}}(r') \geq \Delta_{C_{\text{sym}},\mathcal{B}}(r)$, since essentially the same optimization problem must be solved by the processors in r and r', except that there are possibly more and stronger constraints in r on the choice of INIT_i (corresponding to the smaller bounds for H' in C_{sym}). Thus by Lemma 6.1, we immediately have that $\Delta_C \geq \Delta_{C_{\text{sym}}}$.

To get the inequality in the other direction, we must show that the extra constraints present in C_{sym} do not help in the optimization problem. We do this by showing that for every run r of $\mathcal{B}$ in C, there is a run r' of $\mathcal{B}$ in C_{sym} with essentially the same information. Given r, we define r' so that $\text{INIT}_i(r') = \text{INIT}_i(r)$ and $\delta_{ij}(r) = \delta_{ij}(r')$ if $H'(p_i, p_j) = H(p_i, p_j)$, otherwise we take $\delta_{ij}(r') = H(p_j, p_i) - \delta_{ji}(r)$. Note that if $H'(p_i, p_j) \neq H(p_i, p_j)$, then, by construction, $H'(p_i, p_j) = H(p_j, p_i)$, so this is a legal choice of message transmission times for r'.

Again, let τ_{ij}, τ'_{ij} be the quantities computed in step 2 of $\mathcal{B}$ in runs r and r', respectively, and consider the constraints arising in Eq. (*) in runs r and

r'. If $H'(p_i, p_j) = H(p_i, p_j)$, then clearly we have $\tau_{ij} = \tau'_{ij}$, and the constraints involving τ_{ij} and τ'_{ij} are identical. Suppose we have $H'(p_j, p_i) \neq H(p_j, p_i)$ (possibly because $(p_j, p_i) \notin E$). In this case, $H'(p_j, p_i) = H(p_i, p_j)$ and, by construction, $\delta_{ji}(r') = H(p_i, p_j) - \delta_{ij}(r)$. An easy computation now shows that

$$\begin{aligned}\tau'_{ji} &= \mathrm{INIT}_i(r') - \mathrm{INIT}_j(r') + \delta_{ji}(r') \\ &= \mathrm{INIT}_i(r) - \mathrm{INIT}_j(r) + H(p_i, p_j) - \delta_{ij}(r) \\ &= H(p_i, p_j) - \tau_{ij}.\end{aligned}$$

Substituting $H(p_i, p_j) - \tau_{ij}$ for τ'_{ji} in the constraint $-\tau'_{ji} \leq \mathrm{INIT}_j - \mathrm{INIT}_i \leq H'(p_j, p_i) - \tau'_{ji}$, and using the fact that $H'(p_j, p_i) = H(p_i, p_j)$ and $\tau_{ij} = \tau'_{ij}$, we get $-\tau'_{ij} \leq \mathrm{INIT}_i - \mathrm{INIT}_j \leq H(p_i, p_j) - \tau'_{ij}$, which is the constraint we already had! Thus, there are fewer constraints in the optimization problem for r' than there are for r, hence $\Delta_{C,\mathcal{B}}(r) \leq \Delta_{C_{\mathrm{sym}},\mathcal{B}}(r')$. Again using Lemma 6.1, we get $\Delta_C \leq \Delta_{C_{\mathrm{sym}}}$. ■

Finally, we show that the imprecision of an undirected connected network C is equal to the imprecision of a corresponding completely connected network C_{comp}. Given $C = (G, H, L)$, with $G = (V, E)$ and $L(p, q) = 0$ for $(p, q) \in E$, let $C_{\mathrm{comp}} = (G', H', L')$, where $G' = (V, E')$, $E' = V \times V$ (so G' is a completely connected network on the nodes in V), $L'(p, q) = 0$ for $(p, q) \in E'$, and $H'(p, q) = \min\{\sum_{i=1}^{k-1} H(p_i, p_{i+1}) \mid p_1, \ldots, p_k$ is a path from p to q in $G\}$. Thus the uncertainty in message transmission time between p and q in C_{comp} is the min of the sum of the uncertainties over all paths from p to q in G.

PROPOSITION 6.4. $\Delta_{C_{\mathrm{comp}}} = \Delta_C$.

Proof. The proof is very similar to the proof of the previous two propositions, so we just sketch the details here. For any run r of $\mathcal{B}$ in C_{comp}, let r' be the corresponding run of $\mathcal{B}$ in C. Since there are more and possibly stronger constraints on the choice of the INIT_i's in the optimization problem in r', we must have $\Delta_{C,\mathcal{B}}(r') \geq \Delta_{C_{\mathrm{comp}},\mathcal{B}}(r)$. As before, it follows that $\Delta_C \geq \Delta_{C_{\mathrm{comp}}}$.

For the opposite inequality, suppose r is a run of $\mathcal{B}$ in C. Let r' be the run such that $\mathrm{INIT}_i(r) = \mathrm{INIT}_i(r')$ and $\delta_{ij}(r')$ is $\sum_{k=1}^{m-1} \delta_{k(k+1)}$, where the sum is taken over a path $\pi = q_1, \ldots, q_m$ from p_i to p_j such that the sum of the uncertainties over π is minimal over all paths from p_i to p_j in G. It is straightforward to show that $\Delta_{C,\mathcal{B}}(r) \leq \Delta_{C_{\mathrm{comp}},\mathcal{B}}(r')$, again by showing that the new constraints imposed by this choice of $\delta_{ij}(r')$ do not add any extra information. We leave details to the reader. The argument is again completed by an application of Lemma 6.1. ■

By combining the results of Propositions 6.2, 6.3, and 6.4, we get

THEOREM 6.5. *For any communication network C, we can effectively find a communication network $C' = (G, H, L)$, where G is a completely connected undirected graph, $L(p, q) = 0$, and $H(p, q) = H(q, p)$ for all edges in (p, q) in G, such that $\Delta_{C'} = \Delta_C$.*

7. OPTIMAL PRECISION IN THE PRESENCE OF FAULTS

In order to be practical, an algorithm that achieves a high degree of simultaneity must also be able to tolerate faults. In Section 2 we gave a definition of the essential temporal imprecision inherent in a particular algorithm $\mathcal{A}$ under the assumption that there were no faults. We now extend this definition to allow faults. We consider both processor and communication link faults, and assume that faults are *Byzantine:* faulty processors can maliciously try to sabotage an algorithm, and faulty communication links can deliver a message at any time after it is sent, totally ignoring the bounds given by the H and L functions. We say that *a set F of faults does not disconnect a network C* if for all processors p and q not in F, there exists a path consisting of processors and links not in F from p to q.

Define

$$\Delta_{C,\mathcal{A}}(f) = \max_{p,p',r}\{\Delta_{C,\mathcal{A}}(p, p', r) \mid \text{at most } f \text{ faults occur during run } r, \text{ and these faults do not disconnect } C\},$$

$$\Delta_C(f) = \min_{\mathcal{A}}\{\Delta_{C,\mathcal{A}}(f)\}.$$

Since in Section 2 we assumed there were no faults, $\Delta_{C,\mathcal{A}}$ as defined in Section 2 is equivalent to $\Delta_{C,\mathcal{A}}(0)$. We say an algorithm achieves *bounded precision in the presence of f faults* if $\Delta_{C,\mathcal{A}}(f) < \infty$.

One might expect that since a fault-tolerant algorithm must protect against the possibility of faults even when none actually occur, in the fault-free case it would not perform as well as an optimal non-fault-tolerant algorithm. Surprisingly enough, as far as precision goes, this is not the case. Define the *loss of precision* in algorithm $\mathcal{A}$, denoted $l(\mathcal{A})$, to be $\Delta_{C,\mathcal{A}}(0) - \Delta_C$. Thus $l(\mathcal{A})$ measures by how much the precision achieved by $\mathcal{A}$ differs from the optimal precision achievable if there are no faults.

THEOREM 7.1. *There exists an algorithm $\mathcal{A}$ that achieves bounded precision in the presence of arbitrarily many faults such that $l(\mathcal{A}) = 0$.*

Proof. The required algorithm $\mathcal{A}$ with $l(\mathcal{A}) = 0$ proceeds in two phases. The first phase gets all logical clocks synchronized to within some bound B that depends only on the network C, while the second phase simulates the optimal algorithm $\mathcal{B}$ of Section 4 if there are no faults, and guarantees some bound on the precision if there are faults. We make use of the following key

observation: in the optimal precision algorithm presented in Section 4, if clocks are initially at most B apart, then there is bound to be a solution to Eq. (*) such that $0 \leq \text{SHIFT}_i \leq B$.

The first phase proceeds as follows. When a processor "wakes up" (either because its physical clock reaches a certain reading or becuase it receives a message from another processor) it sets its logical clock to 0 (by appropriately setting TAR) and sends a message to all its neighbors in the network, telling them to wake up. All further phase 1 messages are ignored. Within at most $(n-1)b$ time, where n is the number of processors in the network and b is an upper bound on message transmission time between any pair of processors, the wake up message has diffused to every correct processor in the network. Thus logical clocks are within $B = (n-1)b$ after this phase. Note this is true no matter how many faults there are, as long as the faults do not disconnect the network.

In phase 2, we just simulate the optimal algorithm $\mathcal{B}$ of Section 2, with the following minor changes. In step 2, logical clocks are used instead of physical clocks to measure τ_{ij}. In step 4, processor p_1 tries to compute a solution to Eq. (*) with $0 \leq \text{SHIFT}_i \leq B$. If no such solution exists (this is possible in the presence of faulty processors since such processors may send incorrect values of τ_{ij} to p_1), then p_1 takes $\text{SHIFT}_i = 0$ for all i. In step 5, if the value of SHIFT_j received by p_j is between 0 and B, then p_j sets $\text{TAR}_j := \text{TAR}_j + \text{SHIFT}_j$; otherwise, TAR_j is not changed (this may happen if p_1 is faulty).

It is now easy to check that if there are no faults, then optimal precision is achieved; the proof is the same as that in Section 4. On the other hand, if there are faults, since logical clocks are not shifted by more than B in phase 2 and they are within B at the beginning of phase 2, they can be no more than $2B$ apart after the algorithm terminates, even in the presence of arbitrarily many faults. ■

Theorem 7.1 tells us that there is an algorithm that both achieves bounded precision in the presence of faults and optimizes for the fault-free case. This may prove useful in networks where faults are a relatively rare occurrence. We remark that, unlike many Byzantine agreement algorithms with high fault tolerance, the algorithm of Theorem 7.1 does *not* require authentication.

It is interesting to compare this result to those of Coan *et al.* (1985). In their model there is no uncertainty in message delivery time between correct processors (all messages are delivered in one round). The question there is whether there are fault-tolerant algorithms that guarantee that the processors will perform a given action simultaneously. Answers are given in terms of the number and type of faults. Here we have only shown that there is an algorithm that achieves both bounded precision in the presence of arbitrarily many faults, and optimal precision if there are no faults. It would be interesting to extend the ideas of Section 4 to try to find an algorithm $\mathcal{A}$ such that $\Delta_{C,\mathcal{A}}(f) = \Delta_C(f)$ for $f > 0$.

8. Conclusions and Open Questions

We have given an algorithm that allows processors to achieve optimal precision in arbitrary networks with no clock drift and have given an algorithm to compute the essential temporal imprecision of a communication network. However, a number of questions of both theoretical and practical significance remain open. We list a few of them here:

1. All our results thus far have been proved for the case where physical clocks run at the rate of real time. A more realistic assumption is that physical clocks may drift away from real time, but by at most a bounded rate. More precisely, this means there is a constant $R > 1$ such that

$$R^{-1}(u - v) \leq D(u) - D(v) \leq R(u - v).$$

In practice, this bound is quite small, so that our results for the case of no drift provide a good indication to what can be done even with drift. Nevertheless, it would be interesting to precisely compute the effects of drift. We remark that with drift results such as Proposition 6.2 no longer hold; the actual message transmission time matters, as well as the uncertainty.

2. Our optimal algorithm for precision requires $O(e)$ messages, where e is the number of edges in the network. Can we do better?

3. Our optimal algorithm is highly centralized. One processor calculates the values of SHIFT_i and then sends these values to all the other processors. Clearly decentralization is useful to achieve true fault tolerance. But even in the absence of faults, decentralization may speed up the real-time performance of the algorithm, especially if the computation of the SHIFT_i can be parallelized in a useful way. Can this be done?

4. It would be interesting to minimize the real time required to run the algorithm, particularly when the rate of drift is a nontrivial factor.

5. As remarked in the last section, more work also needs to be done to obtain algorithms that achieve optimal precision in the presence of faults.

6. What is the precise complexity of computing nature's optimal strategy and the essential temporal imprecision? We have shown that it can be done in exponential time, but conjecture it is NP complete.

7. Although we have given an algorithm for computing the essential temporal imprecision, it would be interesting to obtain precise formulas for the imprecision for a number of graphs that arise in practice. We have given such formulas for two- and three-processor networks, as well as tree-like networks, but the exact imprecision of other networks remains open.

8. We have viewed nature as an adversary that will always choose message transmission times between any pair of processors to be constant in a given run, in order to release as little information as possible. In practice, of course, message transmission times are best viewed as being randomly chosen from a probability distribution. In this case, there is some advantage

in taking a number of readings of message transmission times (the τ_{ij} of the algorithm in Section 4). Finding the best strategy for the processors to follow to achieve optimal precision as a function of the probability distribution on message delivery time remains a completely open problem.

Clearly, there is room for more interesting work in this area!

ACKNOWLEDGMENTS

The first author would like to thank Nancy Lynch and Ray Strong for numerous stimulating conversations. Ray suggested the notion of "loss of precision due to fault tolerance" discussed in Section 7. Flaviu Cristian, Yoram Moses, and Larry Stockmeyer provided a number of useful comments and criticisms.

REFERENCES

COAN, B. A., DOLEV, D., DWORK, C., AND STOCKMEYER, L. (1985), The distributed firing squad problem, *in* "Proceedings, 17th ACM Symposium on the Theory of Computing."

DOLEV, D., HALPERN, J. Y., AND STRONG, H. R. (1984), On the possibility and impossibility of achieving clock synchronization, *in* "Proceedings, 16th ACM Symposium on Theory of Computing," pp. 504–511; IBM RJ 4218; *J. Comput. System Sci.*, to appear.

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

KARP, R. M. (1978), A characterization of the minimum cycle mean in a digraph, *Discrete Math.* **23,** 309–311.

LAMPORT, L., AND MEILLAR-SMITH, P. M. (1984), Byzantine clock synchronization, *in* "Proceedings, 3rd ACM Conference on Principles of Distributed Computing," pp. 68–74.

LAWLER, E., (1976), "Combinatorial Optimization," Holt, Rinehart & Winston, New York.

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

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